



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**DETERMINATION OF HIGH-SPEED MULTIPLE
THREAT USING KALMAN FILTER ANALYSIS OF
MARITIME MOVEMENT**

by

Joseph L. Carnes

June 2015

Thesis Advisor:

Co-Advisor:

Second Reader:

David A. Garren

James W. Scrofani

Steven E. Pilnick

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2015	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE DETERMINATION OF HIGH-SPEED MULTIPLE THREAT USING KALMAN FILTER ANALYSIS OF MARITIME MOVEMENT			5. FUNDING NUMBERS	
6. AUTHOR(S) Joseph L. Carnes			8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) A methodology for automatically detecting a swarm attack in the maritime domain is examined in this thesis. These techniques are based upon feeding data into the Kalman filtering algorithm, which is used in the tracking of moving targets based on simulated radar position measurements. Specifically, the expectation of a location of a given moving vessel based upon the Kalman filtering estimates is used to determine if a strong maneuver is occurring. When a given moving target's motion lies outside of the estimated location zone, additional time is required for the estimated track to synchronize the track with the current measurements for this particular moving target. The proposed use of this algorithm is to provide an ability to monitor the maritime traffic within a given area of regard in order to determine if a high-speed maneuvering surface target swarm attack is occurring. The software for this thesis involved the development and testing of object-oriented source code in MATLAB. This work included the development of an algorithm that monitors all traffic and generates a signal spike when a threat has been initiated. A notional gun system was included in order to permit the calculation of survivability estimates when placed inside a larger Monte Carlo simulation.				
14. SUBJECT TERMS Kalman filter, maritime, swarm threat, HSMST			15. NUMBER OF PAGES 87	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**DETERMINATION OF HIGH-SPEED MULTIPLE THREAT USING KALMAN
FILTER ANALYSIS OF MARITIME MOVEMENT**

Joseph L. Carnes

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2015**

Author: Joseph L. Carnes

Approved by: Dr. David A. Garren
Thesis Advisor

Dr. James W. Scrofani
Co-Advisor

Dr. Steven E. Pilnick
Second Reader

Dr. R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A methodology for automatically detecting a swarm attack in the maritime domain is examined in this thesis. These techniques are based upon feeding data into the Kalman filtering algorithm, which is used in the tracking of moving targets based on simulated radar position measurements. Specifically, the expectation of a location of a given moving vessel based upon the Kalman filtering estimates is used to determine if a strong maneuver is occurring. When a given moving target's motion lies outside of the estimated location zone, additional time is required for the estimated track to synchronize the track with the current measurements for this particular moving target. The proposed use of this algorithm is to provide an ability to monitor the maritime traffic within a given area of regard in order to determine if a high-speed maneuvering surface target swarm attack is occurring. The software for this thesis involved the development and testing of object-oriented source code in MATLAB. This work included the development of an algorithm that monitors all traffic and generates a signal spike when a threat has been initiated. A notional gun system was included in order to permit the calculation of survivability estimates when placed inside a larger Monte Carlo simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	THESIS OBJECTIVE.....	2
B.	RELATED WORK.....	3
II.	BACKGROUND	5
A.	HSMST THREAT.....	5
B.	KALMAN FILTERS	6
III.	KALMAN FILTER	9
IV.	IMPLEMENTATION AND RESULTS	17
A.	MATLAB MODEL.....	17
1.	MATLAB Interface	18
2.	Simulation/Input Parameters	18
B.	MODEL MODIFICATIONS AND RESULTS.....	30
V.	RESULTS	37
VI.	PROPOSED FUTURE WORK	39
	LIST OF REFERENCES	65
	INITIAL DISTRIBUTION LIST	67

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Detail of the Kalman filter process.	12
Figure 2.	Demonstration of Kalman filter.	13
Figure 3.	Examination of un-optimized portion of a Kalman filter.	13
Figure 4.	Kalman filter with intentional change in direction.	14
Figure 5.	Enhancement of Kalman filter prediction during object direction change.	15
Figure 6.	Flowchart diagram of simulation environment.	19
Figure 7.	Starting points of scenario. Ownship, civilians, and threats are represented by the blue circle, blue dots, and red dots, respectively.	20
Figure 8.	Starting points of scenario with actual axes ratio.	20
Figure 9.	Gunfire representation. Ownship, in the blue circle, has destroyed a threat ship, as represented with the red line.	24
Figure 10.	Starting prediction (green marks).	26
Figure 11.	Closeup of Kalman prediction. Blue points represent actual measurements, and green points represent predictions from the previous measurement.	26
Figure 12.	Kalman filter error measurements. This is the error amount in distance from the previous prediction and the current measurement at the time indicated. The red dots are threat tracks, and the blue dots are civilian tracks.	28
Figure 13.	Kalman error measurement at time of swarm attack. The red dots represent threat tracks, and the blue dots represent civilian tracks. The red X represent a threat track that has been identified as such. The error is measured between the measurement predicted from the previous measurement and current measurement.	29
Figure 14.	Full simulation in which the ship survived, as none of the tracks reach the blue circle, which represents ownship.	31
Figure 15.	The Kalman error measurement in the aftermath of an attack.	32
Figure 16.	Failure due to insufficient Kalman deviation. Note that the ship has ample time to fire but does not due to failure to classify the threat ship as such. Upper image is at true ratio.	33
Figure 17.	Close-up of threat that remained undetected. The blue line is the original heading and green is after the attack commences. Note that it does not deviate as severely as other threats pictured.	34
Figure 18.	Catastrophic failure due to reload time increased to 15 seconds. Note the large number of threats that are not destroyed and still arriving when the ship is destroyed.	34
Figure 19.	Simulation altered so threats attack closer to ownship in the middle of the x-axis.	35

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Specifications of NAVAIR HSMST analog vehicle, after [11].	5
Table 2.	MATLAB top level functional organization of the STC algorithm.	18
Table 3.	Model parameters.	19
Table 4.	Initial modeling conditions	30
Table 5.	First set of simulation runs.	33
Table 6.	Threats with center-spaced threats.	36
Table 7.	Threats with no Kalman filter.	36

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The High Speed, Maneuvering Surface Target (HSMST) swarm attack is one of the most examined threats against the U.S. Navy today. The attack uses small, fast, cheap boats in numbers great enough to overwhelm the defenses of a given target. These kinds of attacks are typically used against High Value Assets (HVA), destroyers or larger. While it would be easy to avoid such threats by operating a “Blue Water” navy in deep, open ocean, such luxuries are not the reality for the current maritime force.

The extensive use of foreign ports and need to travel in commercial traffic lanes results in naval ships being among civilian traffic. Navy vessels are the most vulnerable when they are within proximity of this traffic, as the threats can hide in neutral traffic until the time of attack. The ability to quickly determine when a ship is under swarm attack and determine hostile actors increases the survivability of the ship under attack.

The objective of this thesis is to develop a method to automatically detect a swarming event. The focus is on making such a method implementable, rather than theoretical. This is accomplished by setting up a simulation in two parts: the simulated portion and the tracker. Keeping the tracker ignorant of the simulation allows it to be utilized in real-world situations as well as be fine-tuned by improving the data of the simulation. This is accomplished by dynamically creating all of the tracks, both civilian and threat, then inputting the various surface tracks into the tracker function, which interprets all data without knowledge of which track is a civilian and which track is a threat. In this way, evaluation can be conducted without bias.

Each object is monitored independently as an individual track, with each track receiving its own Kalman filter position variables and weighting. This provides the ability to judge whether or not the tracker is predicting the swarm event. The simulation code is embedded inside a shell function that sets its random number and collects the results. The starting variable is adjusted until individual metrics, such as mis-labelling civilians, survivability, and minimum distance to threats can be evaluated in Monte Carlo simulated scenarios or real-world data. As the script stands, each scenario is randomly

generated, but if a corner case is found in which the HVA does not survive, a specific scenario can be re-played by fixing the random number seed.

Fixed test threat scenarios can be used as inputs into the tracker to determine feasibility with real data. All that is required is for the tracker to be fed the information from prerecorded, as opposed to the randomly generated, data.

For this specific scenario, the ship is assumed to be moored and stationary, with potential threats being monitored in the harbor channel. The goal is to identify all actors and determine if threats can be identified before they leave the cover of civilian traffic. From there, a gun system with modifiable parameters has been simulated and is used to destroy the simulated threats. The gun has simulated limitations, such as reload and retargeting. Areas of focus can be determined by the survivability percentages altered by the adjustment of these variables.

Civilian traffic provide a low-level “murmur” of Kalman errors, with small movement and measurement errors introduced, as might be expected in moving seas. This also provides us with the possibility of civilian traffic being mistakenly classified as a threat. While this is an interesting metric to collect, it can almost certainly be mitigated in future iterations with additional logic in the code to filter false alarms with threatening vs. nonthreatening direction determination.

The “spike” generated by the sudden errors in multiple threats is meant to be a visual identification for the purposes of this thesis. A more efficient method of establishing criteria to declare a swarm attack is expected to be the subject of future research. A notional gun system with its own state machine is written in a separate module to allow others to create a more detailed model that produces more realistic survivability simulations.

The results of the randomly generated traffic indicate that this method of threat determination is viable. While both measurement and movement errors result in the occasional mis-labelling of civilian traffic as possible threats, the actual swarm event is clearly detected against the random traffic movement errors, usually by a few orders of magnitude.

It is not feasible to have an automated response connected with the tracker in its present stage of development. It could, however, be easily turned into an automated system to improve early detection of a swarm event and highlight possible hostile actors. In the simulation, this detection occurs before the hostile actors have even left civilian shipping lanes.

Doctrine that calls for a quick response increases survivability on a ship caught in such a predicament. The model is built to simulate response time and react with gunfire. In a realistic response, the “layered defense” currently employed by the Navy allows for more options to deal with such a threat, and these can be added in future modifications of the simulation.

Drastic improvement in survivability resulting from the quick identification of a given swarm threat was shown. The additional time afforded the targeted ship allows it to take out a greater number of hostile actors, which, in turn, improves the survivability of the ship.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My loving wife, Tabitha, has been my cheerleader throughout this entire process, enduring many late nights during which I yelled at my computer, and pushing me through when the lack of progress discouraged me. Without her love and support, I'd still be contemplating an approach, and would not be the success I am. I only hope to show her the same amount of encouragement and love with her challenges.

On a more platonic level, I'd like to send my appreciation and gratitude to all the people who supported me at NPS, but especially to Mrs. Sue Hawthorne, Dr. David Garren, Dr. James W. Scrofani, and Dr. Steven E. Pilnick at the Naval Postgraduate School, who have courageously taken on the challenge of reading an engineer's attempt to explain himself through the medium of words.

And my mother, Mrs. Pauli Carnes, a retired English teacher, who, in addition to helping me proofread the text, also reminded me throughout my math-heavy education that I still need to know to spell reel goodly.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Arguably, the United States Navy's last formal battles against an enemy nation-state of any appreciable size in open water (a so-called "blue water" engagement) took place during the Battle of Leyte Gulf. Since then, the Navy's role has mainly been force projection in support of ground troops. Instead, the Navy now must operate globally in peacetime conditions. This makes them increasingly at risk to asymmetric warfare. Asymmetric Warfare (or 4th Generation Warfare) thrives in a condition in which militants and civilians are interspersed [1]. In such an environment, hostile actors can use civilian actors to blend in, striking when conditions favor them. Suicide/Kamikaze style tactics go beyond the actual damage inflicted and can have a demoralizing effect.

Today's Navy is no longer one that can isolate itself from civilian traffic. Peacetime Rules of Engagement, combined with international cooperation and travel through civilian shipping lanes, means that Navy ships are among civilian ship traffic more often than not. With visits to foreign ports of call, as well as repair/upgrade facilities worldwide, today's Navy needs to be able to react to a possible threat at any given time.

One of the closely examined threats continues to be the High Speed, Maneuvering Surface Target (HSMST) in a swarm configuration [2] [3]. Indeed, there is an exercise, called SWARMEX (SWARM Exercise) dedicated to countering that threat [4]. The CIWS Phalanx system, in wide use today, underwent a costly improvement to the 1B variant in an attempt to counter such a threat [5].

The HSMST threat generally consists of a series of inexpensive, unarmored boats rushing a given target ship. Individually, these boats do not pose a threat, as each can be destroyed handily by the target ship before getting close enough to inflict damage; however, the greater maneuverability, speed, and number of craft in a swarm likely will ensure that at least one threat gets close enough to deliver the necessary destructive elements (e.g., Rocket Propelled Grenade (RPG), missile launcher, or explosive laden for a suicide run). In many cases, it would only take one boat getting through a target ship's

defenses to either seriously cripple or destroy the Navy vessel. In this manner, an attack costing tens of thousands of dollars can inflict damage costing millions.

As an example, a gap study made in September 2008 details a Norwegian NATO exercise in which a High Valued Asset (HVA) was engaged by a swarm of smaller boats that were able to hit and retreat without being detected [3]. This study was conducted in a fjord while the HVA was moving. When it was moored or anchored, the danger to the HVA became more pronounced, as it had no opportunity to easily escape.

In congested waterways, the normal markers for hostile intent (closure rate, erratic maneuvering, proximity, etc.) are also negated as chaotic traffic is constantly moving in multiple directions. Maintaining Situational Awareness (SA) is a problem for both bridge watch standers and the personnel manning the Combat Information Center (CIC) as the number, type and intent of surface vessels quickly becomes overwhelming. [3]

The objective of this thesis is to quantify and assist the warfighter with a detection capability to help identify and narrow down the possible threats, as well as show a swarm event as being detectable from civilian traffic before it emerges from the clutter. While this is not intended to be used by itself, it would help in conjunction with other discriminators to create a “weighting” to identify hostile actors in advance of such an attack. In such a scenario, seconds count toward ownship survivability.

A. THESIS OBJECTIVE

The objective of this thesis is to show the ability to detect a swarming event and identify the hostile actors while threat actors are still among civilian traffic. In this thesis, the focus is specifically on movement patterns with no further knowledge of the target in question. The size, shape, or classification of the actor is not a factor in detection of hostile intent.

The Kalman filter is proposed as the method of choice for object travel prediction. The filter, as well as all the simulated objects and gun system, are all modeled in MATLAB.

B. RELATED WORK

In the pursuit of solving the HSMST issue, focus has typically been placed on counter-systems and fleet readiness, with the emphasis mostly on the creation of weapon-systems that can counter such threats automatically (e.g., Phalanx Mod 2B, ship-mounted Hellfire) [6].

William Shannon wrote a paper detailing the need to develop an anti-swarming doctrine, but the paper mostly focuses on land battles, which have their own specific concerns [7].

A paper written for the Naval Postgraduate School in 2002 by Daniel Cobian detailed the possible use of the Javelin anti-tank missile for ship protection from swarming threats [6].

Lokukaluge P. Perera et al. examined the use of a Kalman filter for maritime detection. Their emphasis was on using the extended Kalman Filter based on curvilinear motion [8].

Stateczny, A and Kazimierski, W. looked at the use of Kalman filtering in maritime tracking but only for the application of fusing multiple sensors [9].

Steven Terjesen looked at the use of the Kalman filter for state estimation in regards to small rigid hull inflatable boats [10]. The use of the Kalman filter in this paper is also in regards to sensor fusion, similar to the Stateczny paper [9].

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

The attempts to solve the HSMST issue, as well as an explanation of the Kalman Filter, are examined in this chapter.

A. HSMST THREAT

A Small Boat Swarmed attack is the application of Asymmetric Warfare in the maritime environment. The term “small boat” is ambiguous, but typically means a boat less than 50 feet long (typically, much shorter), usually either built or rigged to be fast and maneuverable.

In this research, the definition from Naval Air Warfare Center Weapons Division (NAWCWD) at Point Mugu, California, as per its simulated surface targets overview is utilized. The specifications were detailed in the “Seaborne Target Overview” at the 41st Annual NDIA Target’s UAV’s and Range Operations Symposium and Exhibition by Jeffrey L. Blume, from NAWCWD [11], as seen in Table 1. The ships were used in an exercise, called SWARMEX [4] (SWARM Exercise), intended to simulate exactly the kind of swarming attack evaluated here. The most relevant aspect of the boat’s specification is the speed, which is truncated to 20 m/s for the purposes of this simulation [11].

Table 1. Specifications of NAVAIR HSMST analog vehicle, after [11].

Specification	Measurement
Overall Length	7 meters
Beam	3 meters
Light Displacement	2 tons
Maximum speed	45+ knots

The main attacking criterion consist of multiple attackers that overwhelm a larger ship’s multiple defenses, allowing at least one of the small boats to get within range of a Rocket Propelled Grenade or even a suicidal attack. This approach has been used by the Tamil Tigers [12], [13] and the Iranian Navy/Revolutionary Guard [14].

While the Sri Lankan Navy did end up countering the threat posed by the HSMST threat of the Sea Tigers, it did so using Swarm tactics of its own [15]. This tactic might be effective for a smaller nation with limited borders to protect. The United States Navy, with its wide reach and investment in larger ships, would not likely be able to employ such tactics worldwide.

New weapon system research and development to counter the HSMST threat are being considered [16]. This indicates that money and time are being spent on evaluating and countering this threat.

B. KALMAN FILTERS

The Kalman filter is an algorithm used to track an item in linear motion [17]. It can predict the next point in the case of linear motion with a surprisingly small amount of information. It has been used in a wide variety of applications, ranging from radar trackers to seismology [18].

The Kalman filter works by an estimation of an initial state, or “seeded” values. Research can be conducted to determine approximately correct values for realistic movement; however, no matter the starting conditions, the filter corrects itself given enough time.

The procedure in a typical Kalman filter is a two-step process. When initial conditions are set, the filter estimates the correct location for the next step. It also generates/updates an estimate of the accuracy of the prediction, known as either a “covariance matrix” or an “uncertainty matrix.”

A second measurement is then taken. The filter determines the amount of error, then reassesses the uncertainty matrix of the previous estimate. It then uses that measurement to estimate the next state. Depending on the initial values, the predictions can vary drastically until settling in an almost steady-state.

The limits on this model are that all ships being tracked need to be moving in a linear fashion, and the sampling rate needs to be known. Note that the sampling rate can vary but needs to be known to pass to the uncertainty matrix. In the simulation used in

this thesis, the boats are headed toward a fixed point on the border of the shipping lane, and the sampling rate is fixed at once per second.

In a realistic situation, the sampling rate would change based on the speed of the tracked ship in relation to ownship, as well as the variability of the sweep of the maritime radar used. The simulation used assumes synchronous position updates on all tracks on the water.

THIS PAGE INTENTIONALLY LEFT BLANK

III. KALMAN FILTER

One of the main advantages of the Kalman Filter is that it can continue to make accurate predictions, even with noise introduced into the measurements. It can continue to do the same with minor changes in movement, correcting and continuing to predict along new headings/speeds.

The Kalman filter used is expanded from a basic, linear version for two reasons. It allows for variance in velocity and tracking in two dimensions. [19].

The measured location in standard x - y coordinates is:

$$M = \begin{pmatrix} x \\ y \end{pmatrix} \quad (1)$$

wherein x is equal to the measured location with respect to the x -axis and y is equal to the measured location along the y -axis.

The predicted measurements are described by the matrix

$$P = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix}. \quad (2)$$

in terms of the x -component of the velocity (i.e., \dot{x}), and the y -component of the velocity (i.e., \dot{y}). These two velocity components are not needed initially and are auto-populated in algorithm.

The bounds of allows noise allowed is described by the matrix:

$$B = \begin{pmatrix} x_{noise} & 0 \\ 0 & y_{noise} \end{pmatrix}. \quad (3)$$

This matrix determines the maximum noise allowable before corrections must be applied to the predictions, where x_{noise} is the maximum permitted noise in the x -direction and y_{noise} is the maximum permitted noise in the y -direction.

The update-movement matrix is given by:

$$U = \begin{pmatrix} \frac{dt^4}{4} & 0 & \frac{dt^3}{2} & 0 \\ 0 & \frac{dt^4}{4} & 0 & \frac{dt^3}{2} \\ \frac{dt^3}{2} & 0 & dt^2 & 0 \\ 0 & \frac{dt^3}{2} & 0 & dt^2 \end{pmatrix} \cdot P_{noise}^2. \quad (4)$$

This matrix determines the variance of each movement track in relation to one another in both the x and y -direction, accounting for both position and velocity. The dispersion of the “0”s indicate that there is no dependency between x and y , and all tracks can move independently of each other, where dt is the time between sampling points, and P_{noise} is the standard deviation of acceleration in (m/s²). The Covariance matrix C_0 is initialized to be equal to the update-movement matrix.

The state update matrix is given by:

$$S = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (5)$$

This matrix is used to predict the next state from the previous state. Again, as x and y positions and velocity have no bearing on each other, the pattern of “0”s ensures the calculation of one is independent of the other.

The measurement function is described by the matrix:

$$F = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \quad (6)$$

The velocity matrix is given by:

$$V = \begin{pmatrix} \frac{dt^2}{2} \\ \frac{dt^2}{2} \\ dt \\ dt \end{pmatrix}. \quad (7)$$

Each iteration of the Kalman filter is preceded by a new measurement.

The Kalman filter can be expressed as the following steps:

1) Compute the movement prediction matrix by combining the state update matrix and the value of the predicted movement matrix at the previous iteration via:

$$M_{pred} = S * M'_{pred} + V * A_{max}. \quad (8)$$

where A_{max} is the maximum acceleration magnitude. The movement prediction matrix is initialized at the first iteration with the value

$$M'_{pred} = \begin{pmatrix} x_0 \\ y_0 \\ 0 \\ 0 \end{pmatrix}.$$

2) Calculate the covariance matrix using:

$$C = S * C * S' + U. \quad (9)$$

Here, C is initialized to be the same as the update-movement matrix (UMM) but is changed as the equation updates.

The following steps attempt to adjust the predictions of the next state in light of the accuracy of the present measurement.

3) The Kalman gain (G) is calculated from

$$G = C * F' (F * C * F' + B)^{-1}. \quad (10)$$

This updates the gain based on the covariance matrix and maximum noise figure.

4) The movement prediction is computed from

$$M_{pred} = M_{pred} + G * (M - F * M_{pred}). \quad (11)$$

The Movement Prediction is utilized primarily as an input into the next iteration of the Kalman filter. If the Gain increases, that means the variability of the covariance matrix is likewise increased as it is inaccurate.

5) The covariance matrix update is computed according to

$$C = \left(\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - G * F \right) * C. \quad (12)$$

At this point, the estimate of the next point can be determined as well as the current point. The prediction is saved and is compared to the next measurement. This process is detailed in Figure 1.

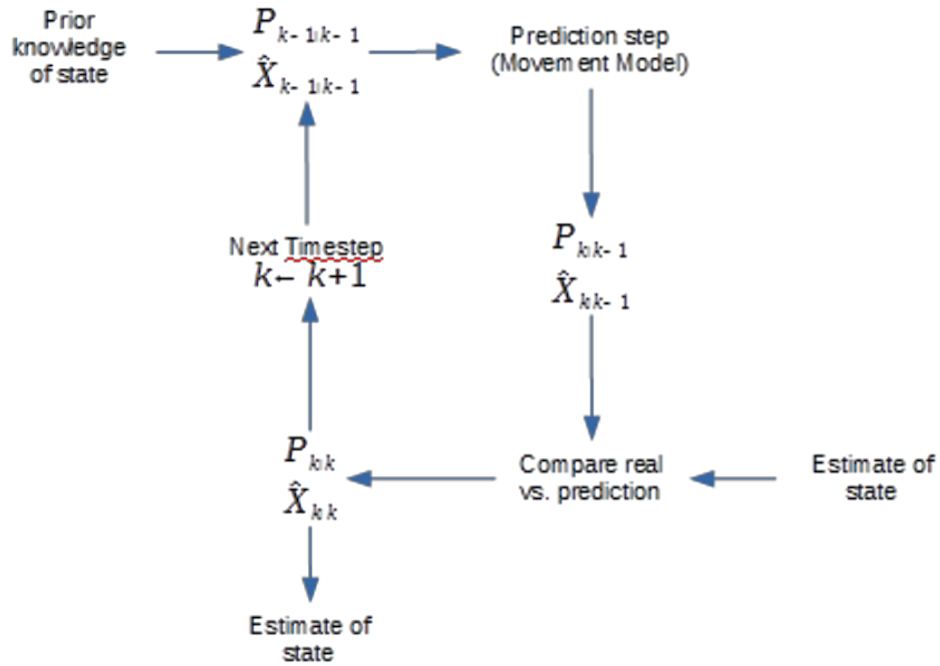


Figure 1. Detail of the Kalman filter process.

A Kalman Filter in operation is shown in Figure 2. The object in motion moves at a consistent pace, without diverting or altering course. Gaussian noise is introduced into both the measurement as well as the motion itself. The error shown at first spikes, as seen in greater detail in Figure 3 (red circles are estimates, green “X”s are measurements). This is because the Kalman filter not optimized in the covariance matrix to anticipate the actual ship motion; however, as the gain is adjusted, the filter prediction becomes more accurate, resulting in a low, flat, error rate.

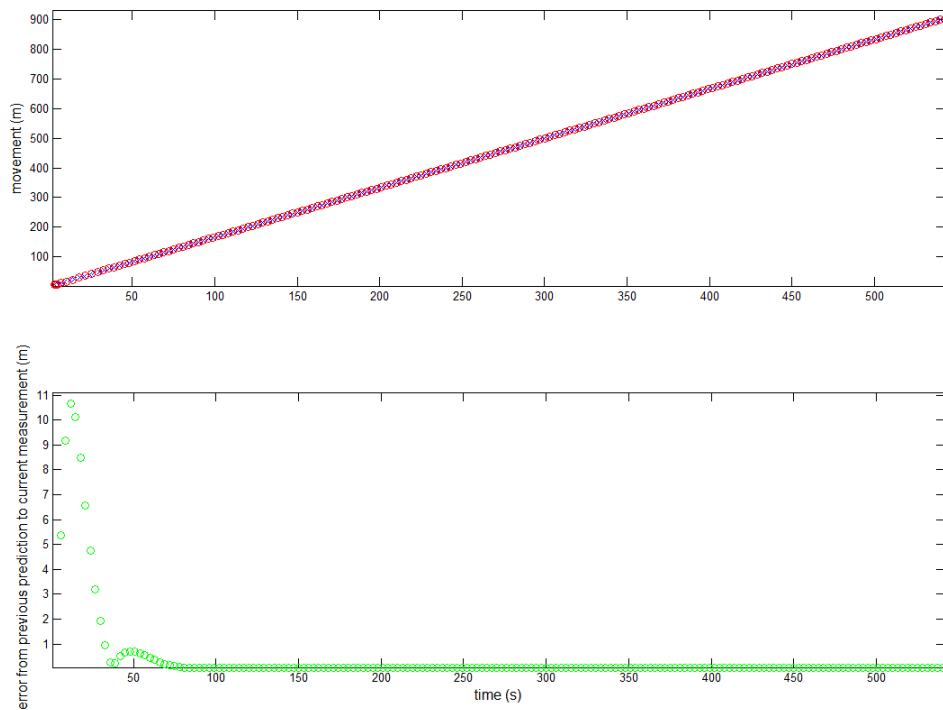


Figure 2. Demonstration of Kalman filter.

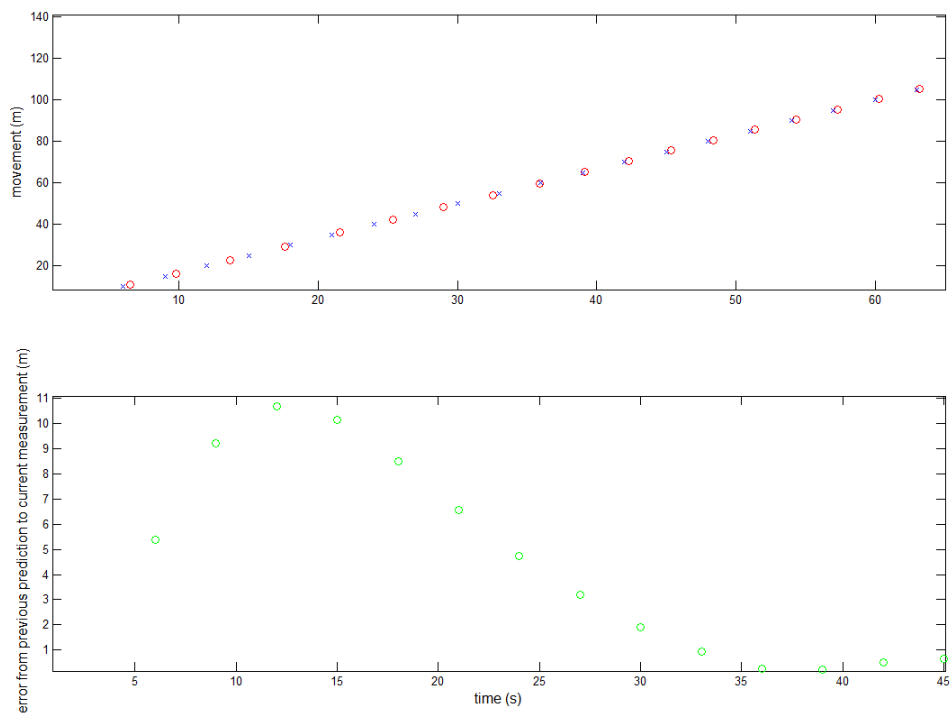


Figure 3. Examination of un-optimized portion of a Kalman filter.

The next modification is to introduce an intentional movement in the course of the object, as seen in Figure 4. The difference between prediction and measurement results in a spike in the error rate, shown in greater detail in Figure 5.

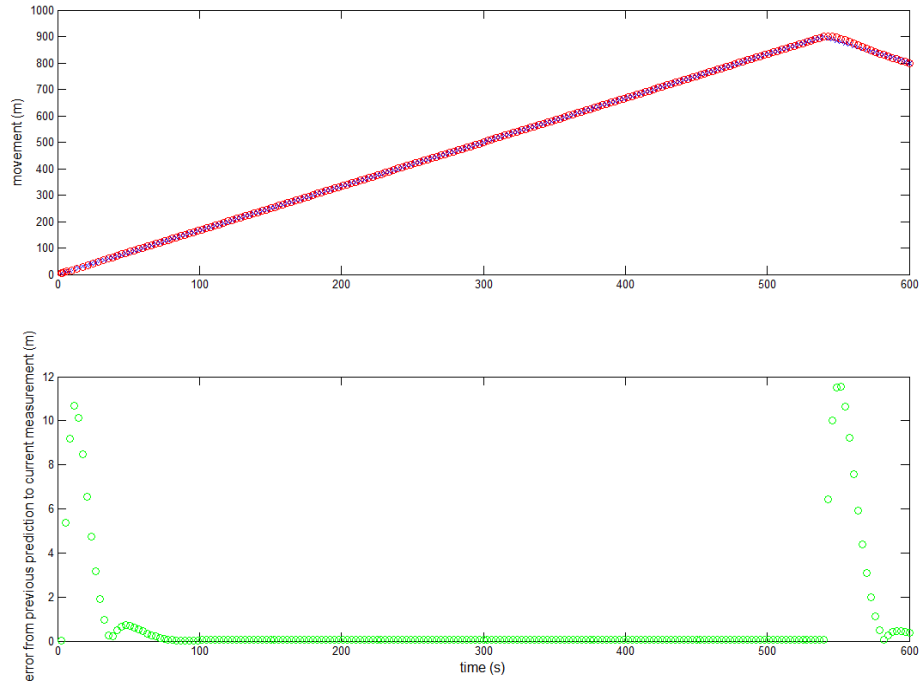


Figure 4. Kalman filter with intentional change in direction.

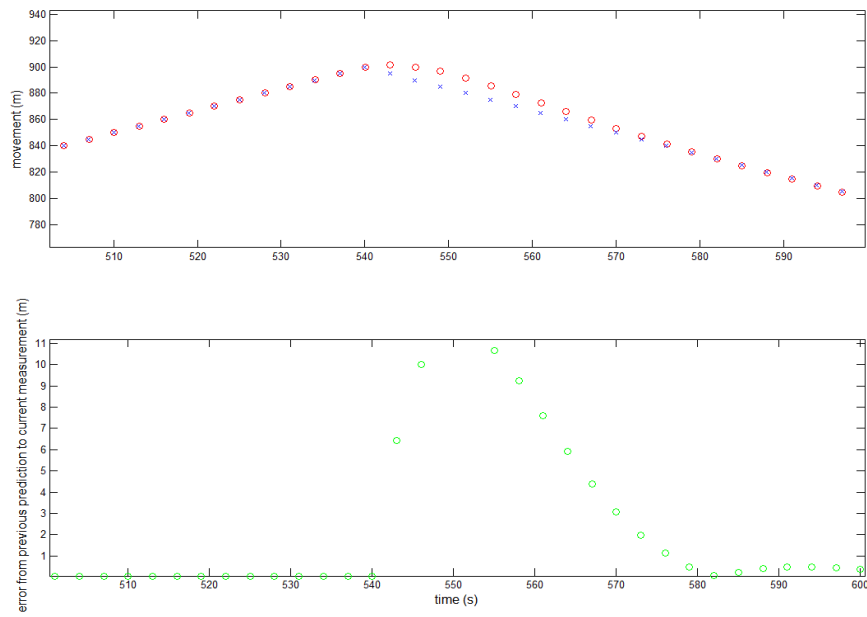


Figure 5. Enhancement of Kalman filter prediction during object direction change.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION AND RESULTS

The Kalman Filter presented in Chapter III is implemented for use in this chapter. MATLAB was used as the programming environment and test bed. Additionally, the MATLAB code is provided and explained.

A. MATLAB MODEL

The MATLAB environment is divided into four parts for maximum flexibility. The use of Object oriented code was used to allow for dynamic creation and movement in the model, as it could be based on a random number scheme, as opposed to fixed movement patterns. This meant the need to “instantiate” each instance of a civilian or threat as well as the non-simulation aware tracker assigned to each return.

The model utilizes random number seeds provided by the outermost function, called “mainfunction.m.” This feeds the random number seed to be fed to the simulation. This allows for examination of specific scenarios in which peculiar behavior has been found or situations in which the ship does not survive that might expose a specific vulnerability. As detailed later, this exact situation happened in a way not originally foreseen.

The top-most simulation file is labeled “thesis” and contains all the functional code necessary to load threats and civilian tracks as well as a separate tracker to keep and maintain all tracker variables necessary to create and maintain the Kalman filter variables.

The remainder of the files are object definition files with their constructors and destructors. The two files labelled “civilian.m” and “threat.m” keep track of the actual locations of the civilian and threat tracks, respectively. The last file, “tracker.m”, is the file that contains the locations input by the simulation, with measurement and movement errors incorporated. The file is not made aware of the simulation space and treats each track equally. Each instantiation contains the state of the Kalman filter variables.

1. MATLAB Interface

As this model is intended for thesis research and not for an operational implementation, the simulation code is run inside of shell intended to input random number seeds and collect results. To that end, there is no graphical user interface.

2. Simulation/Input Parameters

The input files are provided in Table 2. The beginning parameters are provided in Table 3. The flowchart detailing the simulation process is shown in Figure 6.

Table 2. MATLAB top level functional organization of the STC algorithm.

Function	Overview
mainfunction.m	Launches simulation with a random number seed. This allows specific scenarios to be replayed, either in a loop, or in isolation. This function also records the result of each simulation, and saves it to a csv file.
thesis.m	This sets up the simulation environment, calls the appropriate objects, sets up the logic for the trackers, filters, updates, and feeds the appropriate location data to the tracker, iterates the tracker, and updates the simulation world with the results. Additionally, it models the gun system, registering targets that have been removed from simulation, and updating the tracker accordingly.
civilian.m	Object instantiated and intended to hold location and destination variables for each civilian track in the simulated world.
threat.m	Object instantiated and intended to hold location and destination variables for each threat track in the simulated world.
tracker.m	Object instantiated for each track, kept purposefully separate, where the Kalman filter states for each track are kept. Additionally, the status of each track as to whether it is declared a swarming threat is held here.

Table 3. Model parameters.

Parameter	Value
Ownship starting position	1.5 km, 0.0 km
Total possible Civilian tracks	100–120
Total possible threat tracks	30–50
Possible starting positions of all tracks	In a shipping lane 1.5 km-2.0 km from bottom of grid, -15.0 to +15.0 km port and starboard of grid
Gun System	Ready and loaded. Reload time is set to 5.0 seconds initially.

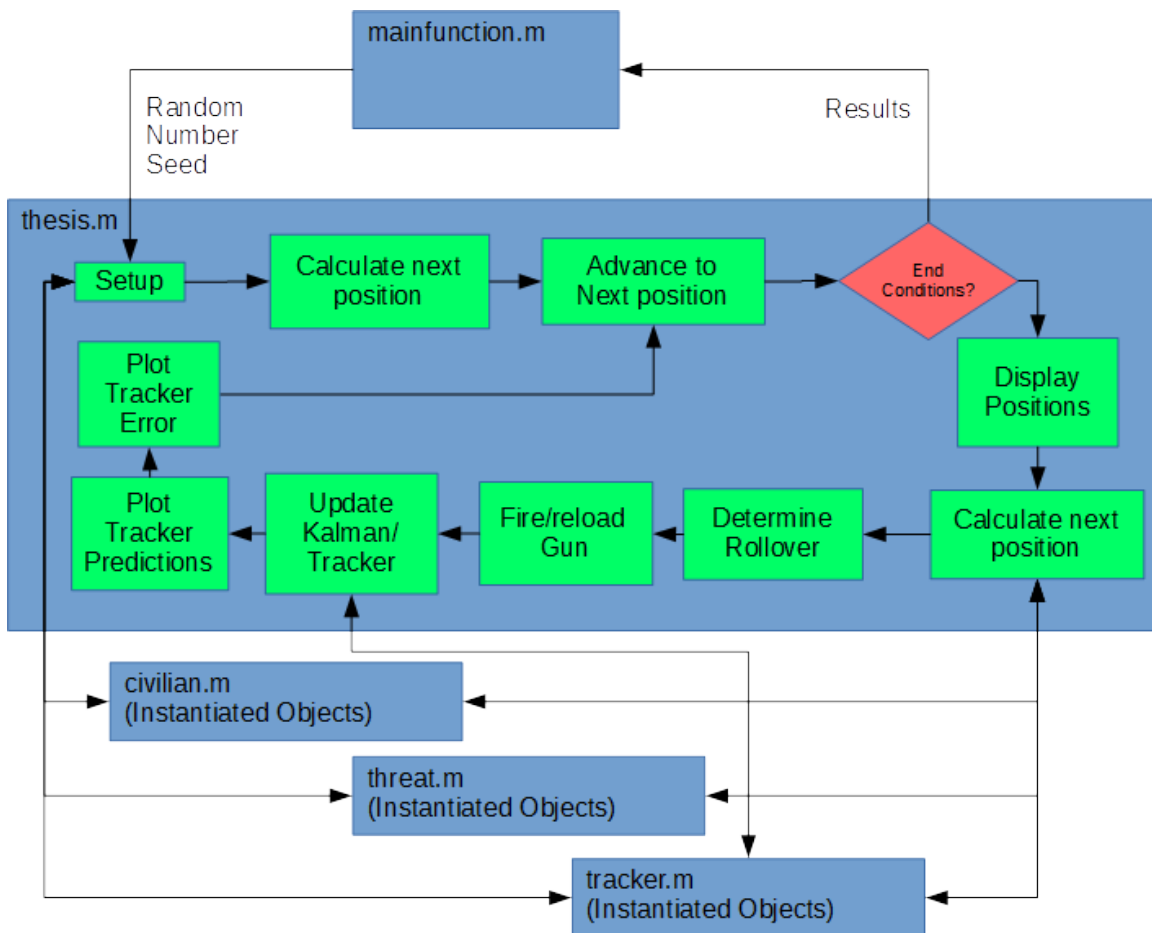


Figure 6. Flowchart diagram of simulation environment.

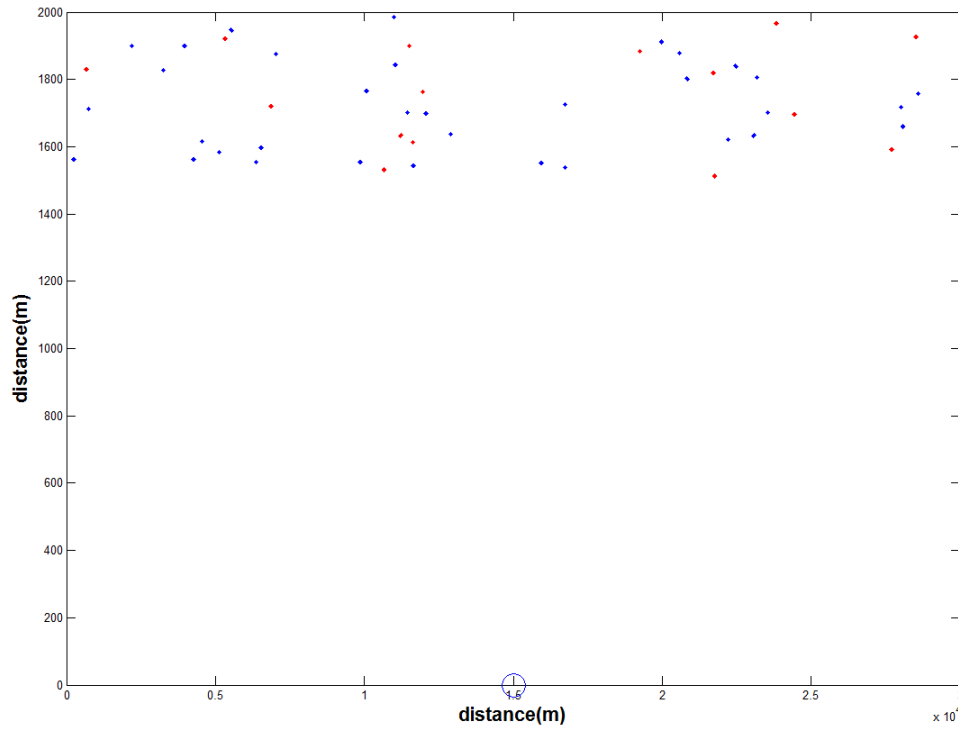


Figure 7. Starting points of scenario. Ownship, civilians, and threats are represented by the blue circle, blue dots, and red dots, respectively.

A possible starting scenario for a given simulation is shown in Figure 7. “Ownship” is shown by the circle at 15.0 km, 0.0 km. The blue and red dots shown are civilian and threat actors, respectively. Note that while they show up in the simulation as being differentiated, the tracker is not informed and only fed positional data.

For the sake of readability, the plots are distorted in axis ratio; however, it is important to remember that the ratio is misleading. In truth, the ratio of axes, when viewed at a 1:1 ratio, is as seen in Figure 8.

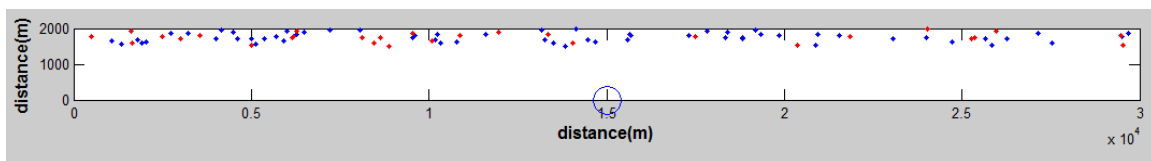


Figure 8. Starting points of scenario with actual axes ratio.

Each simulation is randomized; however, in order to maintain a control on the system, the simulation is fed the random number seed. The simulation is embedded in a slightly smaller program, where the random number seed, the number of threat ships, civilian ships, and whether or not the ship survives are recorded.

When a civilian or threat track is instantiated, it is assigned a randomized position within 15.0 km on either side of the x -axis of ownship, and 1.5 to 2.0 km in the y -axis. A maximum turning radius is assigned in radians, a maximum speed from 10 to 20 m/s, and a random destination point from 1.5 to 2.0 km on the y -axis on either edge (0.0 or 30.0 km in the x -axis). For computational purposes, the next position is initialized to the current position, and the “off edge” flag is set to false (the purpose behind this is detailed later).

The tracker is then fed all the information on the present position of all tracks, both civilian and threat, and assigns each ship instantiation a tracker identification number (trackerid). The reason for this design choice is detailed later. The instantiation of the tracker object also initializes all the Kalman variables, whether or not the track has been “deleted.” The differentiation is that civilian tracks are considered “deleted” if they roll off of the edge, whereas the threat tracks are “deleted” when the gun system destroys them. A “number of hits” counter is also implemented and initialized, which details the maturity of the Kalman filter gain as it approaches steady-state accuracy.

The simulation is based on the assumption that a swarm threat is more effective if the threats are coming from multiple directions, so that when the gun system is implemented, the gun has to retarget in wide arcs to prevent the closest current threat from reaching ownship. To that end, each swarming threat has its destination point immediately re-assigned to an even spacing across the 30.0 km band in the x -axis. This is to avoid a possible random scenario in which all the threats are approaching ownship in a single-file line.

The swarm event in the simulation is also randomly assigned per simulation from 120 to 270 seconds after start of simulation. Each threat is set to arrive at its assigned

waypoint at the time of the swarm event time. As this model does not use unrealistic speeds, the maximum speed for any given threat is capped at 20 m/s.

The next function call is to display the starting points as shown in Figure 7. This function is called in each iteration to update the plot for each positional change.

The next position for each track is then calculated. This is done by finding the direction of the current position and the change in direction needed to go to the next position but capping it at the maximum turning radius called out in the instantiation. This assumes that the tracks run at maximum speed, even during a turn. Future iterations of this code can have the ship slow down to make tighter turns. The reason why the next position was initialized to the present position at instantiation was so that the initial direction change would be zero. This allows the track to start out in the direction needed to point to its assigned destination without falling afoul of the maximum turn radius limitation. Any direction calculated will fall below the maximum turning radius.

At this point, the simulator decides whether or not to assign random movement to the position due to sea movement. In the current code, 5% of the time, a possible 2.0 m shift in movement in both x and y -axis is assigned. This is the random position error, not the measurement error.

The simulator then advances all the tracks to the next position. Note that the tracker is not updated at this point. This is due entirely to the loop mechanics. While this skews initial results, there is a simple method to ensure things get back on track, as detailed later.

The simulator then enters into the main loop, ended only when either ownship is destroyed or when all threat tracks have been eliminated.

The first check of the main loop is to see if the time of the swarm event has arrived. If it has, the only aspect of the simulation that is changed is that all of the threat tracks have their destination positions changed from the edges of the simulation space to ownship.

The tracks are then displayed at their current positions, and the next position is calculated but not advanced.

A check is made on all the civilian tracks to determine if the next movement step exceeds the bounds of the simulation space (threat tracks never exceed the bounds). If it does, the object's "off edge" flag is set to "true." The tracker object assigned tags the track as "deleted." A new civilian track is then created and added to the tracker. The only difference on this created track from the initial instantiations is that the starting position is assigned to one of the boundaries, and the destination is the opposite boundary. This is meant to simulate a ship entering the simulation space, as it cannot realistically appear instantly inside the simulation space. In this way, the number of civilian tracks always remains constant.

At this point, the number of civilian ships is adjusted so that new ships introduced are not skipped when setting up internal loops. There was an attempt to use a built-in function to eliminate tracks when introduced, but unfortunately, it disconnects the trackid correlation.

The gun system is then called. Originally, the gun system was to be unaware of the simulation aspect, but it became unwieldy to program. So, the gun system is sent the tracker array as well as the threat array in the simulation space and the gun state. The gun system evaluates the state of the gun as well as whether any of the tracks have the "taggedasthreat" flag (detailed later) set to "true" but are not destroyed. The gun system then goes through all the threats in such a state and finds the closest one.

The gun system then sets the track array to deleted, the threat array to destroyed, and plots a red line between ownship and the threat that was destroyed, as seen in Figure 9, and advances the gun system's state machine.

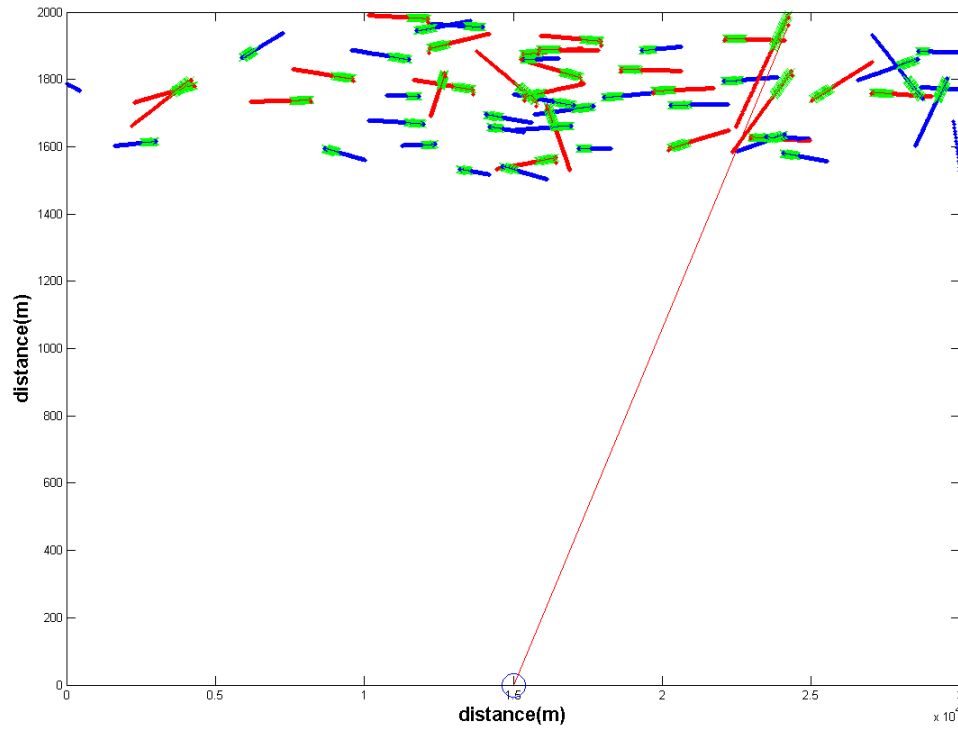


Figure 9. Gunfire representation. Ownship, in the blue circle, has destroyed a threat ship, as represented with the red line.

The gun system state machine is circular, meant to simulate reloading/retargeting, and can be adjusted per simulation expectations. The states are “ready,” “fired,” and a numeric that counts up to five, whereupon the gun returns to the “ready” state. After firing, and upon each evaluation, the gun system is advanced through its state machine. At this time there is no ammunition limit, and all shots are fired as if in the carousel with no delay when loaded in the hoist from the magazine. If desired, future simulations can incorporate other states, such as randomly requiring a second shot on target, delays due to inappropriate rounds loaded in carousel, etc.

All tracks are then evaluated to determine the amount of error from the prediction made at the previous state to the current state. This calculation is only made if the “number of hits” counter exceeds the predetermined number of Kalman gain adjustments

needed to ensure a steady state for the Kalman filter has been achieved. For this simulation's purposes, 70 hits was used.

The tracker is then updated. Both civilian and threat arrays are loaded into the tracker's current state for all the tracks, with the tracker being kept ignorant as to whether or not it is a civilian or threat track that is being updated. The number of hits are incremented for each state. This is necessary so that new contacts coming in from the sides do not clutter the simulation space while the Kalman filter for that particular track is still settling into steady state, as is discussed next.

The next check is made to determine whether or not the minimum amount of hits/time has passed in which to be able to display the tracker predictions. If this is indeed the case, the display function is called and passed the entire tracker array as well as the minimum number of hits necessary to display. The function then evaluates all of the tracker signals to determine if the track is not deleted and has passed the minimum number of hits. If all of those checks pass, it then plots the prediction, as seen in Figure 10 and in detail in Figure 11. Note that in Figure 11, there appears to show a directional bias. This is due to the difference in scale of the x and y -axis, necessary to capture the image and is not the result of a systemic bias.

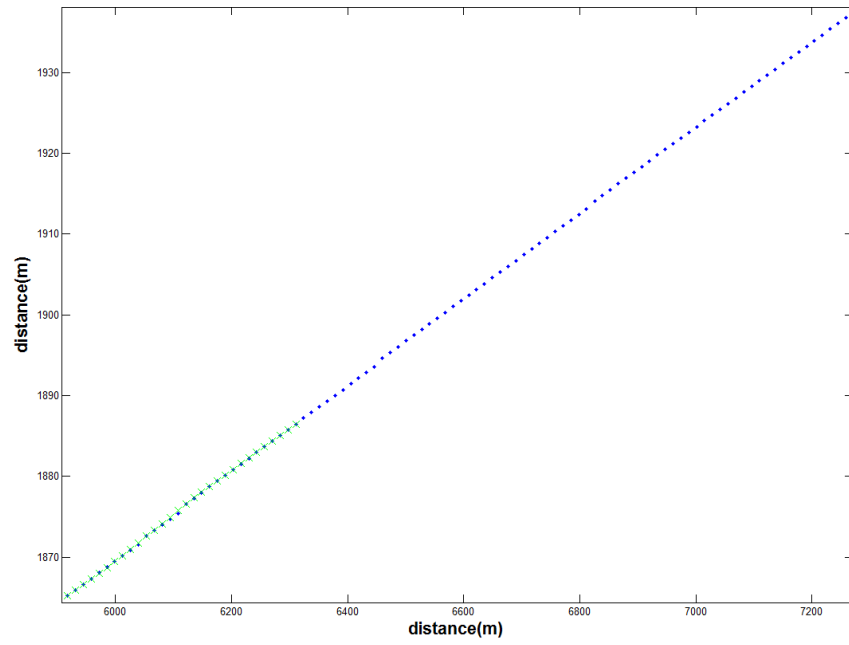


Figure 10. Starting prediction (green marks).

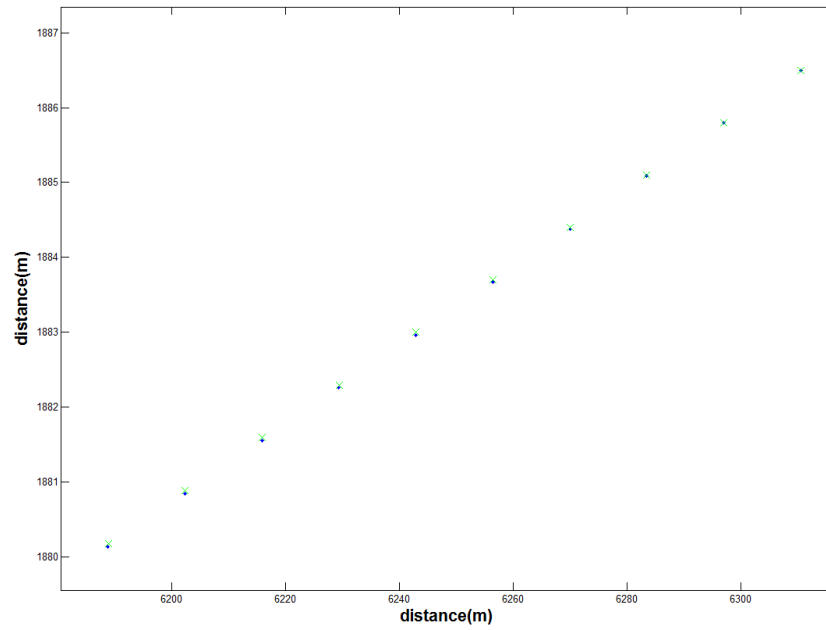


Figure 11. Closeup of Kalman prediction. Blue points represent actual measurements, and green points represent predictions from the previous measurement.

Returning to the main loop, the tracker prediction for the next location is made. The first step is to make sure that the prediction is not being made for a deleted plot. If the plot is not deleted, the tracker then advances the Kalman Filter as was mentioned above. As explained previously, the current location is determined, the estimate is made from the last prediction by comparison to the current location, the gain adjusted, and the covariance estimation for use in the next prediction, with a minor exception code when the measurement is made for the first time.

All arrays and the current time are then fed into an out-of-simulation function that determines what the error is between the current location and the prediction from the last position. Once the 70 samples have been taken, the errors are well into steady state, and a second figure is populated, as seen in Figure 12. When the time of attack is made and a track is declared a target, it is marked with a red “x” as seen in Figure 13. Note there is increased density or error at around 0.0562 meters. This is related to the allowed/assumed noise in the Kalman predictions. When the allowable/assumed noise level is increased, the dense area is expanded and made less dense; however, the errors are minimal and several orders of magnitude lower than the swarm notification looked for.

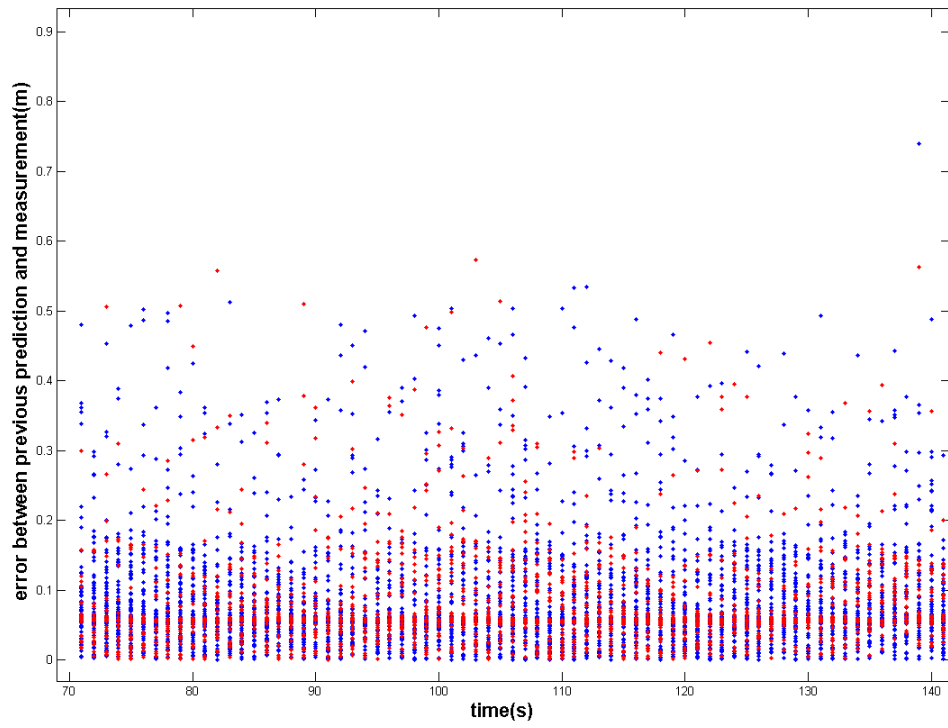


Figure 12. Kalman filter error measurements. This is the error amount in distance from the previous prediction and the current measurement at the time indicated. The red dots are threat tracks, and the blue dots are civilian tracks.

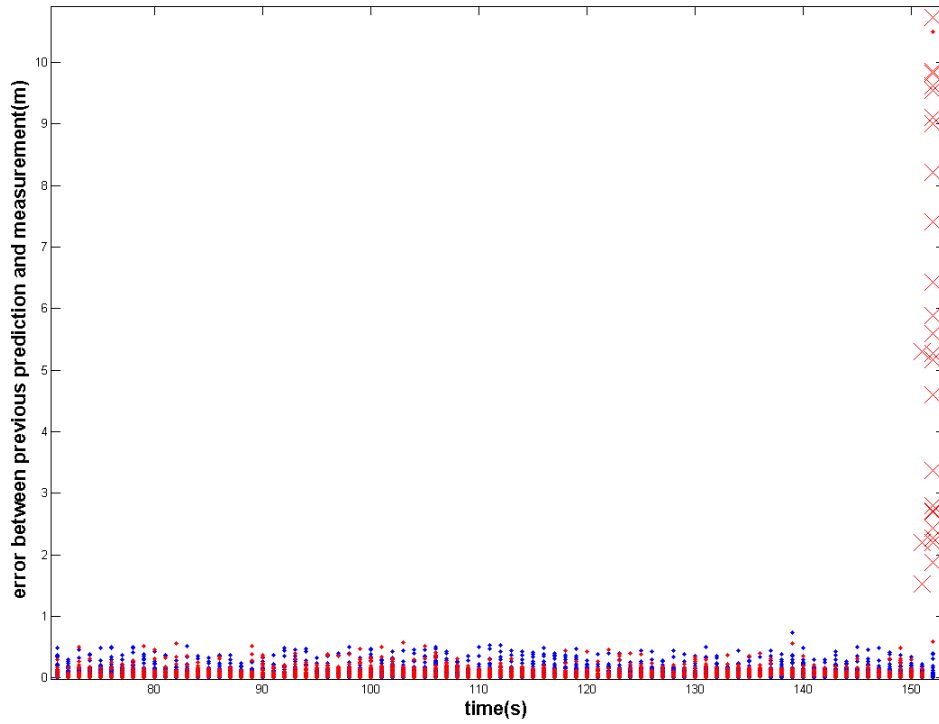


Figure 13. Kalman error measurement at time of swarm attack. The red dots represent threat tracks, and the blue dots represent civilian tracks. The red X represent a threat track that has been identified as such. The error is measured between the measurement predicted from the previous measurement and current measurement.

The ships are then advanced. This is accomplished by making the movement calculated earlier as the next position into the current position. During this check, it is determined whether any of the current tracks will arrive at ownship. Due to the selected geometry, this is accomplished with a simple check to determine if the track has crossed over the x -axis. If it has, ownship is considered destroyed, and one of the two conditions for ending the main loop has been met.

The simulation then makes a determination as to whether or not all threats have been eliminated. If all threats have been eliminated, the other condition for ending the loop has been met, and the loop is ended. If neither condition is met, the loop repeats.

If the loop ends, both screenshots are saved in a time-stamped file, detailing the number of swarm threats, civilian tracks, the number of threats detected, and whether or not the ship survived. The information is passed back to the main function, which records the entire run and records the pertinent data into a Microsoft Excel File.

Please note that due to a MATLAB memory bug, the actual plots are saved to an array and only plotted at the end. This resulted in a drastic reduction in simulation time from hours to seconds.

B. MODEL MODIFICATIONS AND RESULTS

An initial run of 2000 simulations was performed using random number seeds from 1 to 2000. The initial conditions were set per Table 4.

Table 4. Initial modeling conditions

Specification	Measurement
Reload Time	5 seconds
Maximum Speed	20 m/s
Spacing of Threats	Even across x -axis
Number of threats	30–50

The vast majority of the cases resulted in ownship surviving, as seen in Figure 14. The ship is able to fend off multiple attackers without them coming close enough for a suicide attack.

The spike in Figure 15 shows the moment when multiple ships turned from their original heading toward ownship. The dramatic shift is represented clearly against the murmur of ordinary civilian traffic.

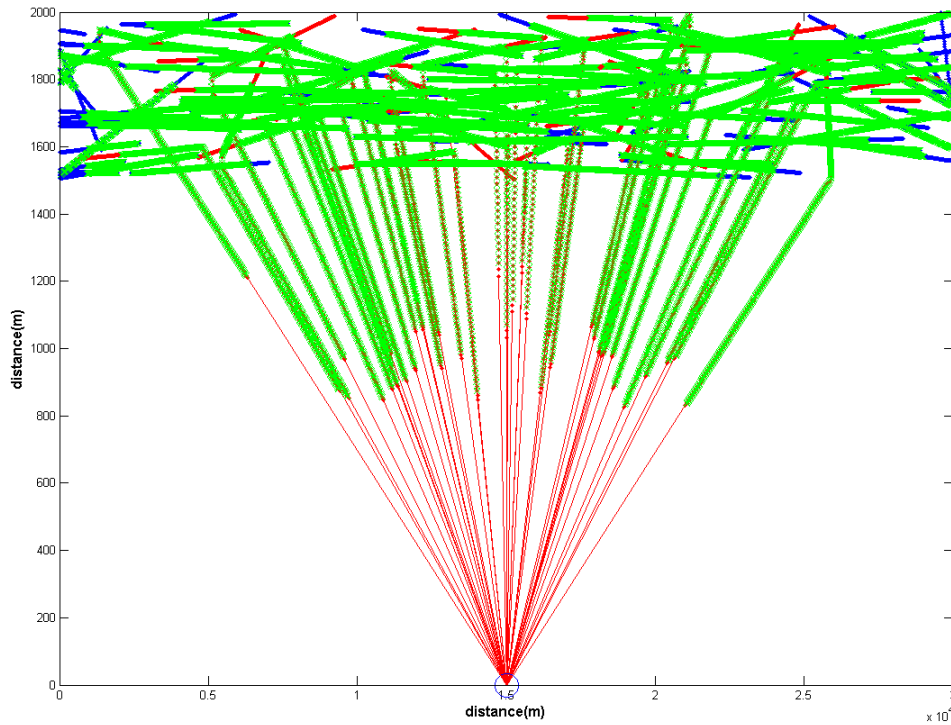


Figure 14. Full simulation in which the ship survived, as none of the tracks reach the blue circle, which represents ownship.

A previously unexamined failure state came to light: in ten out of 2000, or 0.5%, of the simulations, the threat ship does not sufficiently deviate from its initial heading. The Kalman filter error rate therefore does not rise above the threshold, and the threat is able to evade detection, as seen in Figure 16 and Figure 17. In a real-world situation, this would be easily discovered by the crew on duty, but as detection is based entirely on the detection via the Kalman Filter, this is considered a failure.

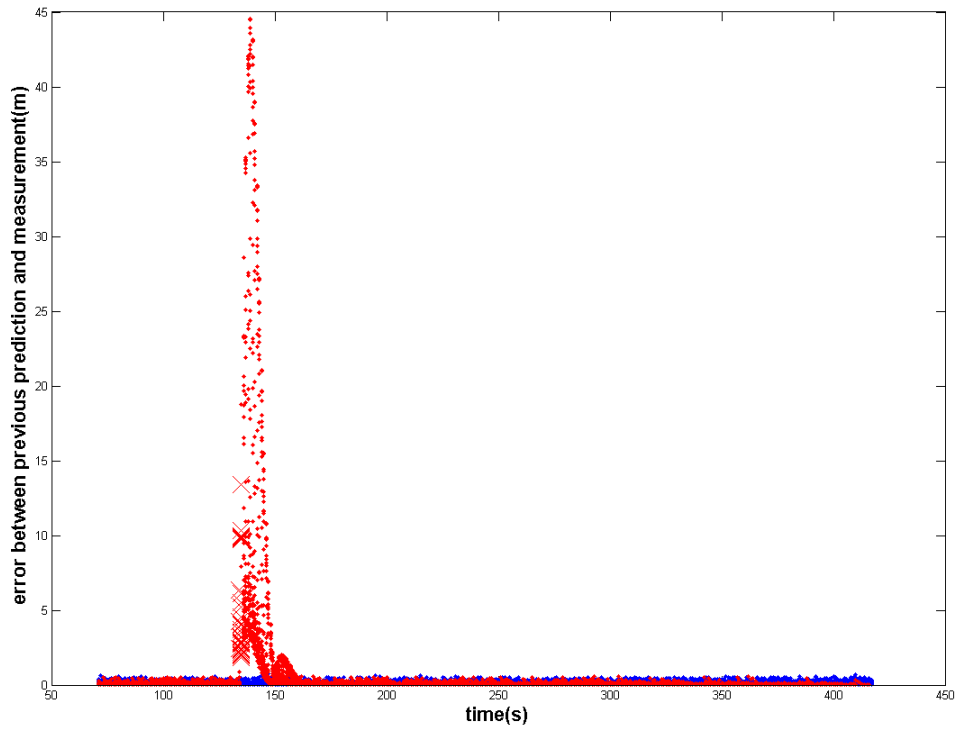


Figure 15. The Kalman error measurement in the aftermath of an attack.

In the next series of simulation runs, the ship's firing time was increased to 15 seconds. This simulates any number of failures, from having to shoot multiple times to the gun system being manually reloaded. For each step, 1000 simulation runs were conducted, and as detailed in both Figure 18 and Table 5, the ship failure rate survivability decreased to 24.4%.

In order to find a middle point at which ownship had a decided advantage but was not succeeding overwhelmingly, the reload time was changed to only double the initial rate, or 10 seconds, as seen in Table 5. This dramatically increased the survivability of the system from 24.4% to 88.7%.

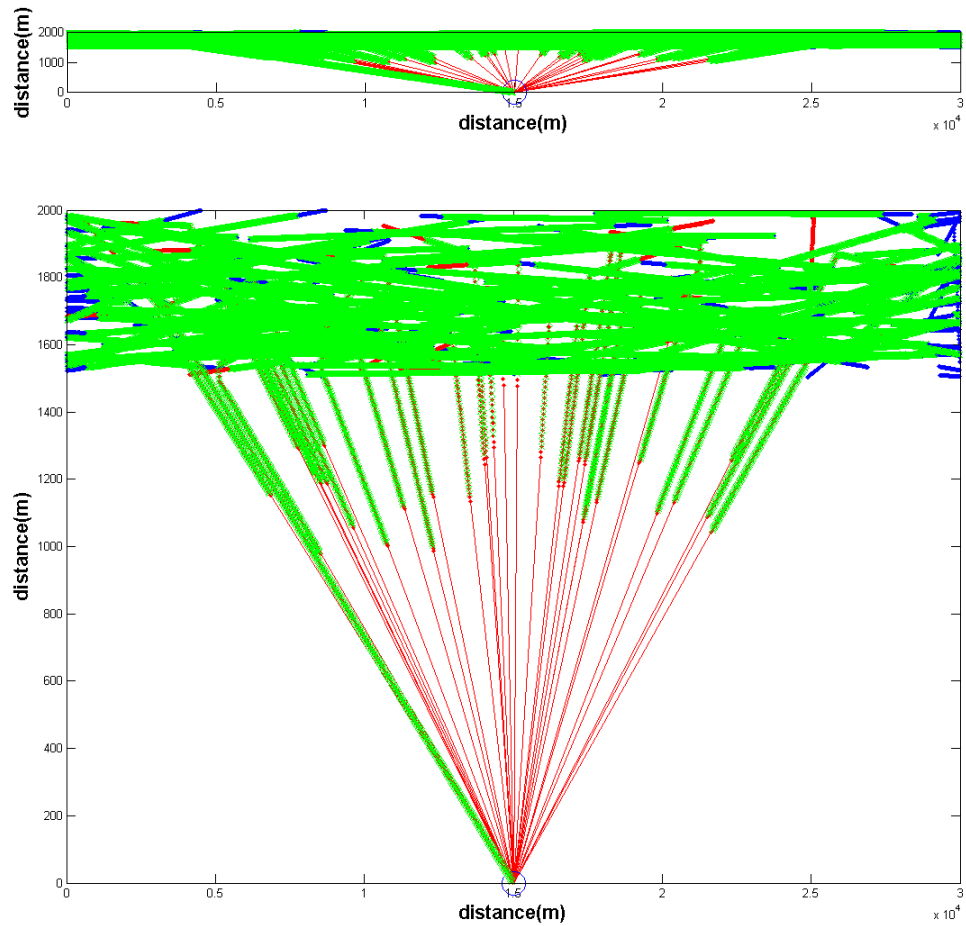


Figure 16. Failure due to insufficient Kalman deviation. Note that the ship has ample time to fire but does not due to failure to classify the threat ship as such. Upper image is at true ratio.

Table 5. First set of simulation runs.

Reload Time	Maximum Speed	Spacing of threats	Number of threats	Survivability
5	20	Even across x -axis	20–40	99.5%
10	20	Even across x -axis	20–40	88.7%
15	20	Even across x -axis	20–40	24.4%
5	20	Even across x -axis	30–50	99%

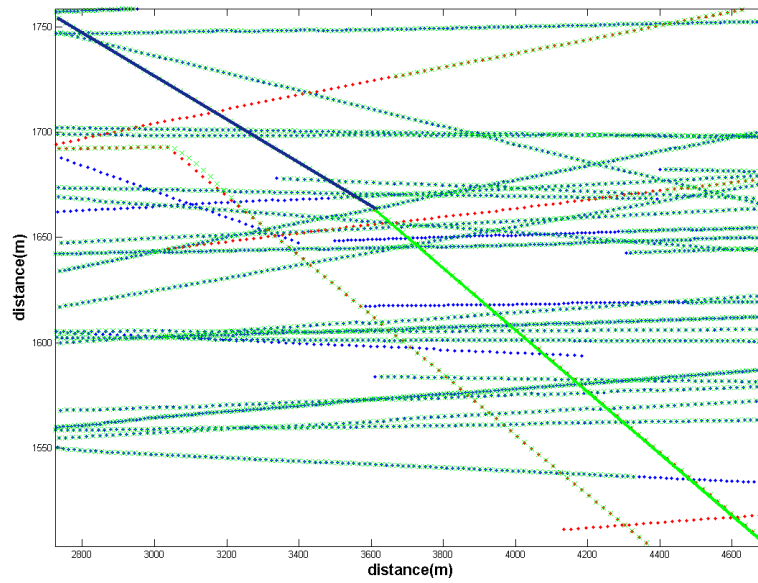


Figure 17. Close-up of threat that remained undetected. The blue line is the original heading and green is after the attack commences. Note that it does not deviate as severely as other threats pictured.

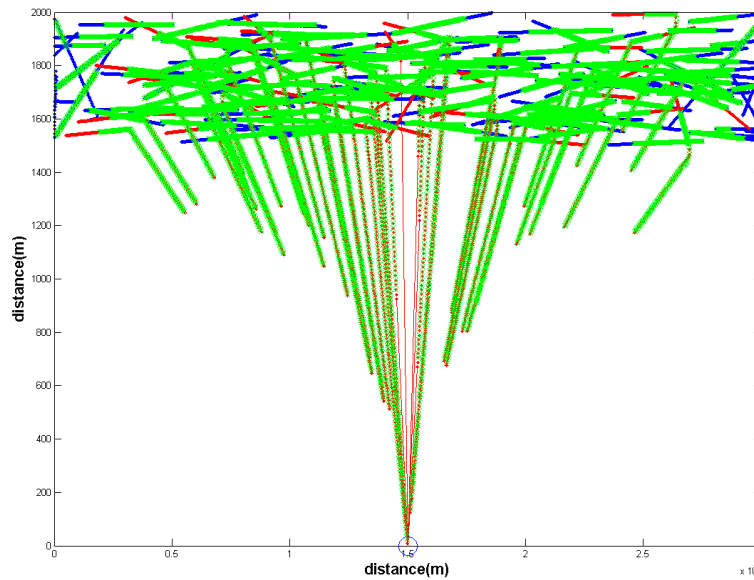


Figure 18. Catastrophic failure due to reload time increased to 15 seconds. Note the large number of threats that are not destroyed and still arriving

when the ship is destroyed.

In an effort to find the worst case, it was then decided to consider the case where all threats attack at the same time from the central point of the shipping channel directly in front of the ship. It was determined that this was providing ownship with a huge advantage, as the ships in the center could be neutralized easily, with more time to take out the ships starting the assault from the edges, which have to travel many times the distance.

The simulation was then altered so that the threat ships attacked from the center. This can be seen in Figure 19. It became quickly apparent that the ship could no longer fend off as many ships as originally thought, as seen in Table 6.

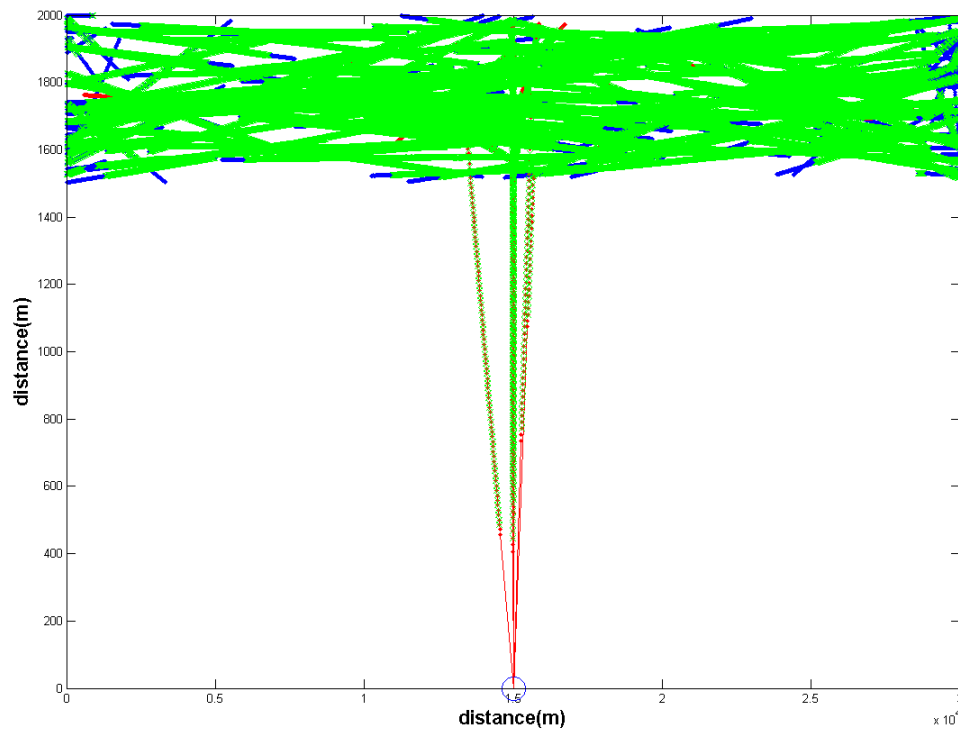


Figure 19. Simulation altered so threats attack closer to ownship in the middle of the x -axis.

Table 6. Threats with center-spaced threats.

Reload Time	Maximum Speed	Spacing of threats	Number of threats	Survivability
7	20	Centerline	20	4.9%
5	20	Centerline	20-40	9.6%
7	20	Centerline	15	63.8%
10	20	Centerline	15	5%
8	20	Centerline	15	27.7%

While these are numbers relative to each other, they only demonstrate survivability with the addition of the Kalman filter. The effectiveness of the Kalman filter itself has yet to be quantified.

To demonstrate that the Kalman filter actually improved survivability, a series of simulations in which the Kalman filter was not activated at all were done. In these demonstrations, the threats are only identified as such when they break free of civilian traffic. The survivability is increased by the time saved from the commencement of the attack to being clear of the shipping channel.

A multitude of simulations were run for comparison, as detailed in Table 7. Survival rates fell across the scenarios by over 40% in some cases.

Table 7. Threats with no Kalman filter.

Reload Time	Maximum Speed	Spacing of threats	Number of threats	Survivability
7	20	Centerline	15	40.8%
10	20	Centerline	20	0%
10	20	Centerline	15	2.4%
7	20	Centerline	20-40	0%

V. RESULTS

The focus of this thesis was to show that the Kalman filter by itself can detect a swarm event, even in the middle of civilian traffic. Further, the model itself shows that survivability is determined by temporal modifications to the model. By that logic, the use of the Kalman filter for detection, if deployed, would enhance the survivability of ownship by alerting the crew to a possible event much faster than would be apparent otherwise.

The framework for the simulation was designed with modularity in mind so that that future work could change aspects of the simulation quickly.

As the Kalman filter involves minor computational requirements, implementation would likely be minimally intrusive and low risk. With today's processing capabilities, if a radar signal processor was designed with modular cards, it is conceivable to have the processing capability with only an additional or replacement processing card.

It should also be noted that the entirety of this testing and discrimination of targets is done without the benefit of any additional sources of information. Additional discriminators would likely reduce false alarms.

As seen in the multiple thousands of runs, there are three main discriminators to determine survivability.

The first discriminator of survivability is the reload time of the gun system. When reload time is increased by seconds, the survivability of the ship changes drastically. This seems to suggest that the correct response to a swarm threat should be to maximize efficiency in terms of target destruction. Having to use multiple shots or spending time re-acquiring after a shot must be decreased through education, drills, and qualification/certification.

The second discriminator of survivability is the starting distance of the threats from ownship. When minimized, ownship was able to handle far fewer hostile actors. If the implementation of this algorithm allows for tighter precision in a given spot, it should be concentrated on the area of the shipping channel closest to ownship.

The third discriminator of survivability is the number of hostile actors. The number of attackers became an issue when they all originated their attack from identical minimal distances from ownship, reducing reaction time.

While reviewing the data, two vulnerabilities to this form of detection became apparent.

The first was discovered by accident, when the ship failed to trip the Kalman error threshold. This was due to changing course at a considerable distance such that only a very slight adjustment was needed. As discussed, in a real-world situation, the crew would likely observe such a tactic from a far distance and be well prepared for a threat once it got within proximity.

The second vulnerability, related to the first, is if the threat is aware of the sensitivity of the Kalman filter, and, rather than turning quickly, performs a slow arc so as to always remain below the threshold. This would require extremely advanced knowledge on the part of the attackers, a fair bit of distance, and could be easily screened out via direction projection of an established track

VI. PROPOSED FUTURE WORK

The future work that can be added on to this model is extensive. While the purpose of this thesis is simply to show that the Kalman filter can be used as a discriminator for a given swarm attack, it can be improved upon and further tested.

The first addition would be the inclusion of direction data as a further discriminator as to determining a ship's intentions. As an example, while a ship that breaks its Kalman filter projection should be investigated, if said ship turns away from ownship, it can be largely discredited as a possible threat until it changes direction again.

With that capability comes the ability to determine if a Navy ship is about to come under attack while inside a given shipping channel rather than outside of it as in the current model. Being able to determine if a given ship is getting closer or farther from ownship, along with the Kalman predictions, would enable a more comprehensive safety barrier.

Factoring in the size of a ship is a powerful discriminator when determining threats. Screening out larger ships or by coming up with a weighting system to largely, but not completely, discredit them as possible threats would quickly knock down the possible attackers to just the smaller vessels, reducing computational load, enabling further analysis.

The concept of a "layered defense" could be implemented, along with an independent modeling system for each gun/missile system. This would provide a more accurate determination as to the survivability of the ship.

Shipping lanes are not always linear. Solving this problem requires that the Kalman Filter be able to "learn" the curves in a given area. This can be accomplished by breaking a given area into small areas and by recording the "normal" movement of ships through those area in terms of maximum curvature and associated direction for each given area. From there, it is relatively simple to determine if a ship matches that allowance. If it deviates from that, it could possibly add to the aforementioned weighting and provide a further indicator of hostile intent. This could make use of previous work by

Lokukaluge and Soares, whose paper is devoted entirely to the use of the Kalman filter to track curved motion [8].

One of the ideas that was abandoned due to time constraints was the idea of a “rolling window” on the detection of events. In a real situation, ships will go over the threshold randomly and could likely get tagged falsely as a threat. One method of screening this error out is to use the swarm event against the attackers by only declaring a swarm event if a number of Kalman Filter error thresholds are exceeded in a given time period. If the number of events does not exceed a certain amount, the system is much less likely to declare a swarm event. Of course, this does come with a problem in that knowledge of this information could make it possible for an enemy to turn sequentially outside of the timing of the window, not tripping the alert as to a swarm event.

One of the assumptions made originally was that the maritime radar had instantaneous feedback on all positions and updated instantaneously. The use of a real maritime radar would likely increase the level of allowable/assumed noise. This would introduce errors in measurement due to Doppler shifts, scan speeds of the radar, etc. The amount of time between samples would change between updates, owing to differences in speed and position of both ownship and external ships.

Another opportunity for future work is to adapt the single-ship swarm detection Kalman filter into a networked sensor system with networked command and control and networked weapons systems such as that envisioned in FORCEnet. In a capstone project for the Naval Postgraduate School in June of 2005 entitled “FORCEnet Implications for a Coalition Maritime Force” [20], a series of simulations were run in which a High Valued Asset (HVA) was surrounded by picket ships.

In the simulation, the HVA was then set upon by missiles, with the picket ships sending present day “track” quality data. This means that the HVA is alerted to the presence of an incoming threat and left to fend for itself with ownship sensors and defenses. The use of missiles provides a high-speed analog for the use of swarming threats, albeit at a much higher speed.

The simulations were then re-run with a simulation of FORCENet in place, sending “targeting” quality data to all ships, which are able to launch interception missiles based on the sensors of other ships, even letting other ships fire when missiles on one platform have become expended.

As shown in this thesis, Kalman filter tracking provides one more advance in survivability against the swarm threat by providing the sailor with advance warning of possible dangers. It could prove to be a crucial information component in the age of asymmetric warfare.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. MATLAB CODE

The MATLAB code used for this thesis is provided below:

First file: “mainfunction.m”:

```
function mainfunction()

%initialize passed back values
totalresult=[];

%set to however many Random Numbers you want.
for i=1:2000
    %display the RNS we are currently displaying in case of a crash
    I

    [civ,thr,result]=thesis(i);
    %record the number of civilian tracks created
    totalresult(i,1)=civ;
    %record the number of threat tracks created
    totalresult(i,2)=thr;
    %record whether or not ownship survived this simulation
    totalresult(i,3)=result;
    totalresult(i)
end

%record the result of the entire simulation run and export as CSV
csvwrite('result.csv',totalresult);

end
```

Second file: "thesis.m"

```
%%
function [numcivshipsstart,numthrships,boom]=thesis(rns)
%clear the screen
clc

%we don't want the random number cleared, but we do want everything
else cleared
clearvars -except rns

%set the random number string
rng(rns);

%randomly set the number of civilian tracks
numcivships=floor(rand*100+20)

%record the number of civilian tracks. This is necessary, as the
number will change with rollovers
numcivshipsstart=numcivships;

%randomly set the number of threat tracks
numthrships=floor(rand*20+30)

%randomly set the timing of attack. Note that it is always greater
than two times the filter settling time.
timeofattack=floor(120+rand()*150)

%Used in earlier versions, abandoned in later ones when we centralized
the attack. This was to ensure that the ships would be evenly spaced
across the x axis
threatspacing=30e3/(numthrships+1);

%Initialize the tracker number counter and the fact that there are
threat ships.
tracknum=0;
threatstillexists=true;

%preallocate the number of civilianships objects in the civilian ships
array, and instantiate/run the constructor.
civilianarray(numcivships)=civilian();

%Do the same for the threat ships
threatarray(numthrships)=threat();

%for the tracker array, instantiate all the tracks for both civilian
and threat ships.
trackerarray(numcivships+numthrships)=tracker();

%Global variable for the minimum number of tracks until we consider the
Kalman filter in "steady state"
mintrackersignals=70;

%Initialization loop for all civilian tracks
for i=1:numcivships
```



```

%iterate the track number
    tracknum=tracknum+1;

%Run the "newship" routine in the object for each new civilian track
    civilianarray(i)=newship(1);

%set the track number into the instantiated civilian track
    civilianarray(i).trackerid=tracknum;

%set the Kalman filter in the TRACK to the first recorded, randomly
generated location
    trackerarray(tracknum).Q(1)=civilianarray(i).currentx;
    trackerarray(tracknum).Q(2)=civilianarray(i).currenty;
end

%Initialization loop for threat ships
for i=1:numthrships
    %iterate the tracknumber. Note that we did not reset the track
numbers for threats
    tracknum=tracknum+1;

    %run the initialization routine in the object
    threatarray(i)=threat;

    %set the spacing target for each threat (note this goes away in the
central version
    threatarray(i).destx=i*threatspacing;

    %set the speed necessary to get to that point by the time the
attack commences.
    threatarray(i).maxspeed=sqrt((threatarray(i).desty-
threatarray(i).currenty)^2+(threatarray(i).destx-
threatarray(i).currentx)^2)/(timeofattack-1);
    %set the track number for the threat ships
    threatarray(i).trackerid=tracknum;

    %Once again, set the associated track's initial Kalman filter
location to the initial settings
    trackerarray(tracknum).Q(1)=threatarray(i).currentx;
    trackerarray(tracknum).Q(2)=threatarray(i).currenty;

    %Cap the speed of each threat.
    if threatarray(i).maxspeed>20;
        threatarray(i).maxspeed=20;
    end
end

%Initialize variables (MATLAB complains if you don't do this)
civdisplay=[];
thrdisplay=[];
trackerpred=[];
civpred=[];
thrpred=[];

```

```

%store initial locations into the display arrays
[civdisplay,thrdisplay]=displayarray(civilianarray,threatarray,numcivships,numthrships,civdisplay,thrdisplay);

%Calculate the next position for civilian and threat tracks
civilianarray=calctime(civilianarray,numcivships);
threatarray=calctime(threatarray,numthrships);

%move the ships (note that "boom" is there just as it is required.)
[civilianarray,boom]=advanceships(civilianarray,numcivships);
[threatarray,boom]=advanceships(threatarray,numthrships);

%Initialize time and gun state
time=0;
gunstate='ready';

%main loop
while (boom==false)&&(threatstillexists==true)
    %increment time, as the main loop doesn't do it
    time=time+1;
    %Check for time equal to attack time
    if time==timeofattack
        %If it is attack time, change each threat ship towards ownship,
        and maximize their speed.
        for i=1:numthrships
            threatarray(i).destx=15e3;
            threatarray(i).desty=0;
            threatarray(i).maxspeed=20;

        end
    end

    %store the position we have advanced to, and calculate the next
    position

    [civdisplay,thrdisplay]=displayarray(civilianarray,threatarray,numcivships,numthrships,civdisplay,thrdisplay);
    civilianarray=calctime(civilianarray,numcivships);
    threatarray=calctime(threatarray,numthrships);

    %determine rollover in next position for civilian tracks, and
    replace if necessary (threat tracks will never get to the edge)

    [civilianarray,trackerarray,tracknum]=determinerollover(civilianarray,trackerarray,tracknum);

    %Renumber (technically, there will be more if a ship rolled over,
    as the number of tracks have increased)
    numcivships=size(civilianarray,2);

    %Advance the state of the gun system, and also fire on threats if
    they exist

    [threatarray,trackerarray,gunstate]=gunsystem(threatarray,trackerarray,gunstate);

```

```

    %update the tracker with new positions

[trackerarray]=updatetracker(civilianarray,threatarray,trackerarray);

    %if the master time is after the minimum time to be able to plot
    predictions, populate the tracker prediction array
    if time>mintrackersignals %screen out early tracking attempts

trackerpred=plottrackerprediction(trackerarray,mintrackersignals,tracke
rpred);
    end

    %make predictions for the next point in all the tracks
    [trackerarray]=trackerprediction(trackerarray);

    %populate the tracker prediction error for the last prediction to
    the current point

[trackerarray,civpred,thrpred]=plottrackererror(civilianarray,threatarr
ay,trackerarray,time,mintrackersignals,civpred,thrpred);

    %Again, advance the ships and determine if the threat ships have
    reached the ships
    [civilianarray,boom]=advanceships(civilianarray,numcivships);
    [threatarray,boom]=advanceships(threatarray,numthrships);

    % Determine if all threats have been eliminated, and set flag
    accordingly
    threatstillexists=determinethreats(threatarray);

end

%Populate the various figures. As detailed, there is a memory leak in
MATLAB's plot function when being updated. So we take the various
arrays that we populated earlier and outputting them into their various
figures.
xaxis=[];
yaxis=[];

for i=1:size(civdisplay,1)
    xaxis(i)=civdisplay(i,1);
    yaxis(i)=civdisplay(i,2);
end

figure(1)
plot(xaxis,yaxis,'.b','MarkerSize',10)
xlabel('distance(m)')
ylabel('distance(m)')
hold on;

xaxis=[];
yaxis=[];

for i=1:size(thrdisplay,1)

```

```

        xaxis(i)=thrdisplay(i,1);
        yaxis(i)=thrdisplay(i,2);
    end

    figure(1)
    plot(xaxis,yaxis, '.r', 'MarkerSize',10)
    hold on;

    xaxis=[];
    yaxis=[];

    for i=1:size(trackerpred,1)
        xaxis(i)=trackerpred(i,1);
        yaxis(i)=trackerpred(i,2);
    end

    figure(1)
    plot(xaxis,yaxis, 'xg', 'MarkerSize',7)
    hold on;

    xaxis=[];
    yaxis=[];

    for i=1:size(civpred,1)
        xaxis(i)=civpred(i,1);
        yaxis(i)=civpred(i,2);
    end

    figure(2)
    plot(xaxis,yaxis, '.b', 'MarkerSize',10)
    hold on

    xaxis=[];
    yaxis=[];

    for i=1:size(thrpred,1)
        xaxis(i)=thrpred(i,1);
        yaxis(i)=thrpred(i,2);
    end

    figure(2)
    plot(xaxis,yaxis, '.r', 'MarkerSize',10)
    xlabel('time(s)')
    ylabel('error between previous prediction and measurement(m)')
    hold on

    filenamebase='';
    clockstamp=clock;

    %Set the current time as the base for the filename to save off the
    figures to
    for i=1:6
        filenamebase=strcat(filenamebase,(num2str(floor(clockstamp(i))))),'-
    ');

```

```

end

% If this number is different than the number of threats initially
generated, we know that the threat was able to succeed due to not
deviating sufficiently when attacking
taggedthreats=0
for i=1:numthrships
    if trackerarray(threatarray(i).trackerid).taggedasthreat==1
        taggedthreats=taggedthreats+1;
    end
end

%Add the initial conditions and the number of threat ships identified
into the filename base.
filenamebase=strcat(filenamebase,'C',num2str(numcivshipsstart),'-
T',num2str(numthrships),'-R',num2str(taggedthreats));

%Add whether or not the ship survived
if boom==1
    filenamebase=strcat(filenamebase,'-boom');
end

%Save off the figures
saveas(figure(1),strcat('screenshots\',filenamebase,'-
simulationspace.png'));
saveas(figure(2),strcat('screenshots\',filenamebase,'-
Kalmanerror.png'));

%clear all variables
close all

end

%%
function
[civdisplay,thrdisplay]=displayarray(civarray,threatarray,numcivships,n
umthrships,civdisplay,thrdisplay)

%master lengths
civlength=size(civdisplay,1);
thrlength=size(thrdisplay,1);
j=1;
for i=1:numcivships

    %offedge calculations have been done. If they are not true, go
ahead and display
    if civarray(i).offedge~=true

        civdisplay(civlength+j,1)=civarray(i).currentx;
        civdisplay(civlength+j,2)=civarray(i).currenty;
        j=j+1;
    end
end
j=1;
for i=1:numthrships

```

```

    if threatarray(i).destroyed~=true

        %Go ahead and display all. They will never be off edge.
        thrdisplay(thrlength+j,1)=threatarray(i).currentx;
        thrdisplay(thrlength+j,2)=threatarray(i).currenty;
        j=j+1;
    end
end

%Display ownship
figure(1)
plot(15e3,0,'o','MarkerSize',20)

end
%%
function passedarray=calctime(passedarray,numships)

%Go through all ships. If they are not over the edge or destroyed,
calculate the next step. This was needed due to the fact that
instantiated objects needed a one-time calculation of this. Rather
than duplicate the functionality, it calls it here.
for i=1:numships
    if (passedarray(i).offedge~=true)&&(passedarray(i).destroyed~=1)
        passedarray(i)=onetimeposcalc(passedarray(i));
    end
end

end

return

end
%%
function [passedarray,boom]=advanceships(passedarray,numships)

%Assume the ship survives.
boom=false;
for i=1:numships

    %a check for offedge and non-destruction
    if (passedarray(i).offedge~=true)&&(passedarray(i).destroyed~=1)

        %If true, advance the ships
        passedarray(i).currentx=passedarray(i).nextx;
        passedarray(i).currenty=passedarray(i).nexty;

        %If the advance takes the ship past the ownship at 0, the ship
has been destroyed.
        if passedarray(i).currenty<0
            boom=true;
        end
    end
end

return
end

```

```

%%
function civship=newship(isnew)

%Instantiate and run the constructor
civship=civilian;

%Only run this section if the ship is replacing a ship rolling off the
edge
if isnew==0

    %when instantiated, they are aimed at one of the edges. This
sequence makes sure that the ship is placed at the opposite edge
    if civship.destx==0
        civship.currentx=30e3;
    else
        civship.currentx=0;
    end

    %Run the next position calculator.
    civship=onetimeposcalc(civship);
end

%Set the previous direction in order to calculate movement and maintain
maximum turn radius calculations
civship.prevdir=finddir((civship.desty-
civship.currenty),(civship.destx-civship.currentx));

end
%%
function enddir=finddir(y,x)

%This function is needed by the next position calculator to determine
the next position's direction, especially in light of maximum turn
radius limitations
%y and x are vectors relating the position to the ultimate destination
of the track in question
if x>0
    enddir=atan(y/x);
else
    enddir=-pi+atan(y/x);
end
return
end
%%
function civatposition=onetimeposcalc(civatposition)

    %Calculate the direction that is needed by the craft
    dir=finddir((civatposition.desty-
civatposition.currenty),(civatposition.destx-civatposition.currentx));

    %Calculate whether or not the direction change is over the maximum
allowable turning rate
    if abs(civatposition.prevdir-dir)>civatposition.turningmax

```

```

        %If indeed the maximum turning rate has been reached, set to the
allowable direction
        if (civatposition.prevdire-dir)>0
            dir=(civatposition.prevdire-civatposition.turningmax);
        else
            dir=(civatposition.prevdire+civatposition.turningmax);
        end
    end;

%Calculate the next position based on the intermediate or final next
step.

civatposition.nexty=civatposition.currenty+sin(dir)*civatposition.maxsp
eed;

civatposition.nextx=civatposition.currentx+cos(dir)*civatposition.maxsp
eed;

%Add some random jostle into the system
%Randomize?
if rand<.05
    range=(rand-.5)*1;
    civatposition.nextx=civatposition.nextx+range;
    civatposition.nexty=civatposition.nexty+range;
end

%Set the previous direction for the next calculations
civatposition.prevdire=dir;

return
end
%%
function [trackarray]=updatetracker(civarray,thrarray,trackarray)

for i=1:size(civarray,2)

    %As long as the track isn't over the edge, update the tracker to
the current position, and increment the number of hits on each track.
    if civarray(i).offedge~=true
        trackarray(civarray(i).trackerid).Q(1)=civarray(i).currentx;
        trackarray(civarray(i).trackerid).Q(2)=civarray(i).currenty;

trackarray(civarray(i).trackerid).numberofhits=trackarray(civarray(i).t
rackerid).numberofhits+1;
    end
end

for i=1:size(thrarray,2)

    %Do the same for threat ships, but don't do it for tracks that have
been destroyed.
    if trackarray(thrarray(i).trackerid).deleted~=true
        trackarray(thrarray(i).trackerid).Q(1)=thrarray(i).currentx;
        trackarray(thrarray(i).trackerid).Q(2)=thrarray(i).currenty;

```



```

trackarray(thrarray(i).trackerid).numberofhits=trackarray(thrarray(i).t
rackerid).numberofhits+1;
    end
end

end
%%
function [endresult]=determinethreats(thrarray)

%This function simply goes through all threat hits, and determines
whether or not all threats have been destroyed.
endresult=false;

for i=1:size(thrarray,2)

    if thrarray(i).destroyed==0;

        %We found one!
        endresult=true;
    end

end

end
%%
function
[civarray,trackarray,trnum]=determinerollover(civarray,trackarray,trnum
)
%This function determines if any of the civilian tracks will roll over
the edge of the simulation, and sets up a replacement track to roll in
at a randomized method.

%Pull the current size of the civilian array.
maxrecord=size(civarray,2);

%All civilian tracks
for i=1:size(civarray,2)

    %we don't want to re-evaluate tracks that have already been
declared off edge.
    if civarray(i).offedge==false

        %if over either edge
        if (civarray(i).nextx<=0) || (civarray(i).nextx>=30e3)    %rolled
off of edge

            %declare it off edge
            civarray(i).offedge=true;

            %Delete it in the tracker
            trackarray(civarray(i).trackerid).deleted=true;

            increment the tracker number

```

```

        trnum=trnum+1;

        %Increment the size of the civilian array
        maxrecord=maxrecord+1;

        %For that new civilian track, instantiate a ship, and run
the constructor
        civarray(maxrecord)=newship(0);

        %set the tracker number in the civilian array
        civarray(maxrecord).trackerid=trnum;

        %Run the onetime position calculator.
        civarray(maxrecord)=onetimeposcalc(civarray(maxrecord));

        %Set the tracker's Kalman filter position to the current
position
        trackarray(trnum).Q(1)=civarray(i).currentx;
        trackarray(trnum).Q(1)=civarray(i).currenty;
    end;
end
end

%%
function [trackarray]=trackerprediction(trackarray)

%In this Tracker-only function, for every track that isn't deleted, run
the Kalman Filter prediction
for i=1:size(trackarray,2)
    if trackarray(i).deleted==false
        trackarray(i)=Kalmanfilter(trackarray(i));
    end
end

end

end
%%
function
trackerpred=plottrackerprediction(trackarray,mintrackersignals,trackerp
red)

%For every track that isn't deleted, and has the minimum number of
hits, take the Kalman Predictions and store in an independent array.

j=1;

trackerlen=size(trackerpred,1);

for i=1:size(trackarray,2)

    if
(trackarray(i).deleted==false)&&(trackarray(i).numberofhits>mintrackers
ignals)

```

```

        trackerpred(trackerlen+j,1)=trackarray(i).Q_estimate(1);
        trackerpred(trackerlen+j,2)=trackarray(i).Q_estimate(2);
        j=j+1;

    end

end

end
%%
function [track]=Kalmanfilter(track)

    % load the given tracking
    track.Q_loc_meas = [ track.Q(1); track.Q(2)];

    % do the Kalman filter

    % Predict next state of the Ship with the last state and predicted
    motion.
    track.Q_estimate = track.A * track.Q_estimate + track.B * track.u;

    %predict next covariance
    track.P = track.A * track.P * track.A' + track.Ex;

    % Kalman Gain
    track.K = track.P*track.C'*inv(track.C*track.P*track.C'+track.Ez);

    % Update the state estimate.
    if ~isnan(track.Q_loc_meas)
        track.Q_estimate = track.Q_estimate + track.K *
(track.Q_loc_meas - track.C * track.Q_estimate);

    end

    % update covariance estimation.
    track.P = (eye(4)-track.K*track.C)*track.P;

end

%%
function
[trarray,civpred,thrpred]=plottrackererror(civarray,thrarray,trarray,ti
me,mintrackersignals,civpred,thrpred)

j=1;

%Get the size of the civilian and threat array prediction arrays
civlen=size(civpred,1);
thrlen=size(thrpred,1);

for i=1:size(civarray,2)

```

%For all civilian tracks that are not off the edge, get the differential between the previous prediction and the current position. Also included are a "reset" if needed.

```

    if (civarray(i).offedge~=true)
        temp=sqrt((civarray(i).currentx-
trarray(civarray(i).trackerid).Q_estimate(1))^2+(civarray(i).currenty-
trarray(civarray(i).trackerid).Q_estimate(2))^2);
        if
((trarray(civarray(i).trackerid).numberofhits)>mintrackersignals)&&(tra
rray(civarray(i).trackerid).deleted==0)
            flux=temp-trarray(civarray(i).trackerid).baseline;
            civpred(civlen+j,1)=time;
            civpred(civlen+j,2)=flux;
            j=j+1;
        else
            trarray(civarray(i).trackerid).baseline=0; %temp;
        end
    end
end

```

```

j=1;

```

%Do the same for the threats

```

for i=1:size(thrarray,2)
    if (thrarray(i).offedge~=true)
        temp=sqrt((thrarray(i).currentx-
trarray(thrarray(i).trackerid).Q_estimate(1))^2+(thrarray(i).currenty-
trarray(thrarray(i).trackerid).Q_estimate(2))^2);
        if
((trarray(thrarray(i).trackerid).numberofhits)>mintrackersignals)&&(tra
rray(thrarray(i).trackerid).deleted==0)
            flux=temp-trarray(thrarray(i).trackerid).baseline;
            if
(flux>1)&&(trarray(thrarray(i).trackerid).taggedasthreat==0)
                trarray(thrarray(i).trackerid).taggedasthreat=1;
                figure(2)
                plot(time,flux,'xr','MarkerSize',20)
                hold on
            else
                thrpred(thrlen+j,1)=time;
                thrpred(thrlen+j,2)=flux;
                j=j+1;
                %figure(2)
                %plot(time,flux,'.r','MarkerSize',10)
                %hold on
            end
        else
            trarray(thrarray(i).trackerid).baseline=0; %temp;
        end
    end
end
end

```

```

end
%%
function

%This advances the gun system through its various stages and fires on
threats.
[thrarray,trarray,gunstate]=gunsystem(thrarray,trarray,gunstate)

%Set a number ludicrously long distance. Also, set the index number.
comparemin=1e6;
indexmin=1;

%If the gun is ready to fire
if strcmp(gunstate,'ready')

    %Go through all threats
    for i=1:size(thrarray,2)

        %Is the target tagged as a threat and not already destroyed?
        if
            (trarray(thrarray(i).trackerid).taggedasthreat==1)&&(thrarray(i).destroyed~=1)

                %Get the distance from this threat to ownship
                temp=sqrt((thrarray(i).currentx-
15e3)^2+(thrarray(i).currenty)^2);

                %If it is closer than other threats, set as the valid
target
                if temp<comparemin
                    comparemin=temp;
                    indexmin=i;
                end
            end
        end

        %If we are not set to the far off distance
        if comparemin~=1e6

            %Mark target as destroyed in both tracker and threat array
            trarray(thrarray(indexmin).trackerid).deleted=true;
            thrarray(indexmin).destroyed=1;

            %Draw the destruction line
            figure(1)
            plot([15e3                                thrarray(indexmin).currentx],[0
thrarray(indexmin).currenty],'-r','Color','r','LineWidth',1)
            hold on;

            %Set the gun state to fired
            gunstate='fired';
        end
    end
end

```

```
%State machine:  fired → countdown → ready, repeat.
if strcmp(gunstate,'fired')
    gunstate=1;
elseif gunstate<5
    gunstate=gunstate+1;
else
    gunstate='ready';
end
end
```

Third file: "civilian.m"

```
classdef civilian
    properties
        currentx
        currenty
        nextx
        nexty
        destx
        desty
        maxspeed
        turningmax
        prevdir
        trackerid

        %Initialized as in the simulation space
        offedge=false;

        %Included because the tracker will declare it destroyed,
        regardless of type
        destroyed=0;

        %To be included in future add-on work. This will be tripped if
        the civilian gets declared a threat.
        falsealert=false;
    end
    methods

        %Civilian constructor
        function civilian=civilian()

            %No restrictions on turning
            civilian.turningmax=2*pi;

            %Set y axis distance randomly
            civilian.currenty=1.5e3+.5e3*rand;

            %For calculation purposes, set the next state to the
            current one.
            civilian.nexty=civilian.currenty;

            %Set destination y axis randomly.
            civilian.desty=1.5e3+.5e3*rand;

            %Randomly set speed
            civilian.maxspeed=10+10*rand;

            %Randomly set x position (overwritten if a roll-off
            replacement)
            civilian.currentx=rand*30e3;

            %Randomly set x axis destination
            if rand>.5
                civilian.destx=0;
            else
                civilian.destx=30e3;
            end
        end
    end
end
```

```
        end

        %Once again, set for calculation purposes
        civilian.nextx=civilian.currentx;
    end

end

end
```


Fourth file: “threat.m”

```
classdef threat
    properties
        currentx
        currenty
        nextx
        nexty
        destx
        desty
        maxspeed
        turningmax
        prevdir
        trackerid

        %Threat isn't destroyed yet!
        destroyed=0;

        %Needed for compatibility with other tracker functions
        offedge=false;
    end
    methods
        function threat=threat()

            %Threat ships can't turn as fast!
            threat.turningmax=pi/4;

            %Random y axis placement
            threat.currenty=1.5e3+.5e3*rand;

            %Needed for calculation reasons
            threat.nexty=threat.currenty;

            %Destination pre attack-time
            threat.desty=1.5e3+.5e3*rand;

            %Set the max speed (will be overwritten immediately
            afterwards)
            threat.maxspeed=10+10*rand;

            %Randomly set x axis location
            threat.currentx=rand*30e3;

            %Randomly set pre-attack destinations
            if rand>.5
                threat.destx=0;
            else
                threat.destx=30e3;
            end

            %Randomly set current x axis location
            threat.nextx=threat.currentx;
        end
    end
end
```

Fifth file: "tracker.m"

```
classdef tracker
    properties
        offedge=false;
        error
        baseline
        deleted=false;

        %record number of hits, only display after the wild errors are
removed
        numberofhits=0;
        taggedasthreat=0;

        % maximum acceleration magnitude
        u = .005;

        %First two are immediately changed to the first location
settings upon instantiation
        Q= [0;0; 0; 0];

        %estimate of initial location estimation of where the target is
(what we are updating)
        Q_estimate = [0;0; 0; 0];

        %Maximum noise in x and y direction
        xnoise = 1;
        ynoise = 1;

        %Maximum noise in matrix form. I was trying to put the noise
into this form after a variable define, but Matlab didn't like it.
        Ez = [1 0; 0 1];

        %These are the update movement model matrix and covariance
matrix model. For the purposes of MATLAB, I needed to define them
separate, but the Covariance is initialized to be the same as the
other. The (.1) at the end is the movement noise
        Ex = [1/4 0 1/2 0; ...
        0 1/4 0 1/2; ...
        1/2 0 1 0; ...
        0 1/2 0 1].*(.1)^2;
        P = [1/4 0 1/2 0; ...
        0 1/4 0 1/2; ...
        1/2 0 1 0; ...
        0 1/2 0 1].*(.1)^2;

        % (Coefficient matrices):

        %State Update Matrix
        A = [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1];

        %Velocity Matrix
        B = [(1/2); (1/2); 1; 1];

        %Measurement Function
        C = [1 0 0 0; 0 1 0 0];
```

```

        %Initialization of the Kalman Gain
        K = [];

        %Location measurement instantiation.
        Q_loc_meas
    end

    methods

        %There so there is a constructor method, but nothing is
        declared.
        function tracker=tracker()

            end

        end
    end
end

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] W. S. Lind, "Understanding fourth generation war," *Military Review*, pp. 12–16, September-October 2004.
- [2] J. Arquilla and D. Ronfeldt, *Swarming and the Future of Conflict*. Santa Monica, CA: RAND Corporation, 2000.
- [3] A. Tiwari, "Small boat and swarm defense: A gap study." M.S. thesis, Dept Modeling, Virtual Env., and Simulation, Naval Postgraduate School, 2008.
- [4] J. Handwork, "NSA-PC brings SWARMEX to successful close," Department of the Navy, [Online]. Available: http://www.navy.mil/submit/display.asp?story_id=12464. [Accessed 03 May 2015].
- [5] Raytheon. "Phalanx close-in weapon system." [Online]. Available: <http://www.raytheon.com/capabilities/products/phalanx/>. [Accessed 13 May 2015].
- [6] D. Cobian, "Sea Javeline: An analysis of naval force protection alternatives." M.A. thesis, Grad. Sch. Bus. & Pub. Policy, Naval Postgraduate School, 2002.
- [7] W. D. Shannon, "Swarm tactics and the doctrinal void: Lessons from the Chechen wars." M.S. thesis, Dept. Nat'l. Sec. Affairs, Naval Postgraduate School, 2008.
- [8] L. P. Perara and C. G. Soares, "Ocean vessel trajectory estimation and prediction based on extended Kalman filter," in *The Second International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2010.
- [9] A. Stateczny and W. Kazimierski, "A concept of decentralized fusion of maritime radar targets with multisensor Kalman filter," Vilnius, Lithuania, 16–18 June 2010.
- [10] L. S. Terjesen, "Navigation system design and state estimation for a small Rigid Hull Inflatable Boat (RHIB)." M.S. thesis, Dept. Mech. & Aerospace Eng., Naval Postgraduate School, 2014.
- [11] P. Jeffrey L. Plume, "U.S. Navy seaborne targets—New directions in a time of change," in *47th Annual NDIA Targets, UAV's and Range Operations Symposium and Exhibition*, 2003.
- [12] A. Fuard. "Sea Tigers adopting suicide attack tactics," *Sunday Times Newspaper*, 17 February 2008. [Online]. Available: <http://www.sundaytimes.lk/080217/News/news0021.html>. [Accessed 13 May 2014].

- [13] Ministry of Defence, “Humanitarian Operation Factual Analysis July 2006-May 2009,” Democratic Socialist Republic of Sri Lanka, 2011.
- [14] F. Haghsheenas. “Iran's doctrine of asymmetric naval warfare,” The Washington Institute, 21 December 2006. [Online]. Available: <http://www.washingtoninstitute.org/policy-analysis/view/irans-doctrine-of-asymmetric-naval-warfare>. [Accessed 13 May 2015].
- [15] Jane's Navy International, “Sri Lanka learns to counter Sea Tigers' swarm tactics,” Jane's Navy International, pp. 20–25, 2009.
- [16] M. B. Deitchman, “Opportunities in power projection and integrated defense,” in *ONR Naval S&T Partnership Conference and ASNE Expo*, 2012.
- [17] R. E. Kálmán, “A new approach to linear filtering and prediction problems,” *Research Institute for Advanced Study*, vol. Series D, no. 82, p. 12, 1960.
- [18] N. D. Crump, “A Kalman Filter approach to the deconvolution of seismic signals.” Society of Exploration Geophysicists, 1972.
- [19] “Object tracking with Kalman filter! (in amazing 2-d!).” Student Dave's Tutorials, [Online]. Available: <http://studentdavestutorials.weebly.com/object-tracking-2d-kalman-filter.html>. [Accessed May 2014].
- [20] J. Carnes, C. Coy, B. Dantimo, V. Davila, M. Edwards, R. Fong, J. Hill, A. Hyder, B. Kokes, J. Lim, J. Lomeli, C. Smith and H. Wang, “FORCEnet implications for a coalition maritime force,” Monterey, CA: Naval Postgraduate School, June 2006

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California